Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Simulación de redes de colas para la validación de sistemas de procesado en tiempo real

TITULACIÓN: Grado en Ingeniería Telemática

AUTOR: Felipe Boix Grandío

DIRECTORA: Esther Salamí San Juan

FECHA: 30 de marzo del 2017

Título: Simulación de redes de colas para la validación de sistemas de procesado en tiempo real

Autor: Felipe Boix Grandío

Directora: Esther Salamí San Juan

Fecha: 30 de marzo del 2017

Resumen

En estos últimos años se está incrementando considerablemente la utilización de aviones no tripulados (UAVs o drones), principalmente para tareas de teledetección. En la mayoría de los casos, el UAV adquiere datos durante el vuelo y posteriormente se realiza un post-procesado, que puede tardar entre uno o dos días, antes de generar el resultado final. Para algunas aplicaciones, como pueden ser las aplicaciones de soporte a incendios forestales o salvamento marítimo, este procedimiento no es aceptable, ya que se requiere una respuesta prácticamente inmediata. En estos casos, es posible equipar el UAV con un sistema procesador que ejecute los algoritmos de procesado a bordo y genere el resultado en tiempo real.

Dentro de este contexto, se plantea la necesidad de planificar, para cada misión, cuál es la máxima tasa de adquisición de datos a la que puede trabajar el sistema en función de los distintos algoritmos involucrados y de los recursos (o núcleos de procesamiento) disponibles, y que asegure un buen rendimiento.

Un sistema multiprocesador se puede interpretar como una red de recursos compartidos que dan servicio a un número elevado de clientes. Gracias a la teoría de colas, se nos proporciona una base teórica que permite analizar el grado de servicio que se puede dar en el sistema. En los modelos donde tanto la distribución del tiempo entre llegadas como el tiempo de servicio siguen una distribución exponencial, el sistema se puede resolver de forma analítica. Sin embargo, en un caso más general, donde los tiempos siguen distribuciones distintas de la exponencial, puede ser necesario recurrir a técnicas de simulación para analizar el sistema.

Hoy en día, la simulación por eventos discretos es una herramienta fundamental para diseñar y desarrollar todo tipo de sistemas. Mediante la simulación se puede llegar a sacar conclusiones óptimas para después ejecutarse en un caso real.

En el caso que nos ocupa, tenemos un sistema de colas en el que las imágenes actúan como clientes entrantes que necesitan un servicio. Los servicios los proporciona la misma ejecución de los algoritmos en los núcleos de procesamiento.

Esta investigación se centra en ver de qué manera se comportan las colas, dependiendo de cómo se modele el sistema. Uno de los parámetros en los que nos centraremos durante el proyecto es el tiempo medio de espera en cola. Un tiempo medio de espera en cola pequeño asegura que las colas permanezcan dentro de límites manejables, pero un valor demasiado pequeño nos puede llevar a una utilización poco eficiente de los recursos. Por el contrario, un tiempo medio de espera en cola grande indica un número de clientes en espera grande, y que llegado a un cierto límite, hace que el sistema no sea estable.

Para ello, se realiza en primer lugar un estudio del comportamiento de cinco algoritmos de procesado utilizando distribuciones exponenciales. El sistema se ha resuelto de las dos maneras: de forma analítica (utilizando el lenguaje de programación R), y mediante técnicas de simulación (utilizando el entorno de trabajo OMNeT++), con el objetivo de confirmar que los valores simulados establecen cierta concordancia con los valores teóricos, y de esta manera validar el entorno de simulación utilizado durante el proyecto.

No obstante, si decidimos actuar en un caso real, se observa que tanto el tiempo entre llegadas como el tiempo de ejecución de los algoritmos siguen distribuciones que se alejan bastante de la exponencial. Esto quiere decir que una planificación basada en el modelo exponencial puede resultar demasiado conservadora, impidiendo sacar el máximo provecho del sistema a la hora de procesar las imágenes. Es por ello, que se deben ajustar los parámetros de la distribución de probabilidad del tiempo de servicio, esto permitirá simular cada algoritmo con la distribución que más se le ajuste, permitiendo ser más realistas y a la vez agresivos en la planificación.

Una vez realizado este ajuste, se volverán a realizar las simulaciones imputando a los servidores de cada algoritmo la distribución elegida. Esto permitirá obtener la tasa máxima de llegadas de cada algoritmo para después decidir cuál de todas será la que límite el sistema.

Finalmente, se aplicarán los resultados obtenidos a dos casos de uso. Por un lado, se implementará una red formada por cuatro algoritmos, cuyo fin es localizar puntos calientes en áreas afectadas por el fuego, ya apagado, pero que pueden ser susceptibles de generar un nuevo incendio. En el segundo caso, la misión consiste en un solo algoritmo cuyo objetivo es detectar bancos de medusas.

Title: Simulation of queuing networks for the validation of real-time processing systems

Author: Felipe Boix Grandío

Director: Esther Salamí San Juan

Date: March 30 th 2017

Overview

In recent years, the use of Unmanned Aerial Vehicles (UAVs or drones) has increased considerably, mainly for remote sensing tasks. In most cases, the UAV collects data during the flight; these data are processed afterwards, during the post-processing phase that may take between one and two days, before generating the final result. In some applications, such as support tasks for forest fires or sea rescue, this delay is not acceptable. In these cases, it is possible to equip the UAV with a processor system which launches the processing algorithms on board and generates the result in real time.

Given this context, it appears the necessity of scheduling, for each mission, what is the maximum data acquisition rate in which the system can work according to the different algorithms involved as well as the resources (or processing cores) available, insuring the proper performance of the system.

A multiprocessing system can be interpreted as a net of shared resources that provide service to a large number of customers. The queueing theory provides the basis that allows for the analysis of the service degree that can be given in the system. For those models where the distribution of the inter-arrival time and the service time is exponential, the system can be resolved in an analytical way. However, in a more general case where the times follow distributions different from the exponential, it may be necessary to appeal to simulation techniques to analyze the system.

Nowadays, the simulation by discrete events is an essential tool for designing and developing all kinds of systems. Through this simulation it is possible to get optimum conclusions that afterwards can be executed in a real case.

In our case, there is a queue system in which the images act as incoming customers who request a service. The services are provided by the execution of the algorithms in the processing cores. This research is focused on analyzing how the queues behave, depending on how the system is modeled. For the evaluation, we will focus on the average waiting time in queue. A small average waiting time in queue guarantees the queues remaining within manageable limits. Yet, a very small value may lead to an inefficient use of

resources. By contrast, a big average waiting time in queue indicates a large number of customers waiting that, when reaching the limit, makes the system unstable.

To carry out this, we begin by studying the behavior of five processing algorithms using exponential distributions. The system has been resolved in two ways: analytical (using the R programming language) and through simulation techniques (using the work environment OMNeT++), with the aim of proving that the simulated values establish a certain level of concordance with the theoretical values and thus, validating the simulation environment used during the project.

Nevertheless, if a real case is evaluated, it is possible to observe that both the inter-arrival time and the execution time of the algorithms follow a distribution far different from the exponential. This means that a scheduling based on the exponential model may result too conservative, avoiding getting the maximum profit from the system when processing the images. For this reason, the parameters of the time-of-service probability distribution must be fitted. It will allow assigning each algorithm the distribution which best fits, allowing then, a more aggressive scheduling and thus, being able to process the maximum number of images per time unit.

Once the adjustment has been done, the simulations will be launched once again setting the chosen distribution of each algorithm in the servers. This will allow obtaining the maximum incoming rate of the algorithm with the aim of deciding which rate will define the system limits.

Finally, the results obtained will be applied in two use cases. On the one hand, a network composed by four algorithms will be implemented in order to locate the hot spots in areas affected by the fire. On the other hand, in the second case, the mission consists of a single algorithm with the aim of detecting jellyfish banks.

Agradecimientos

En primer lugar, quiero agradecer a la directora del proyecto Esther Salamí su soporte y compromiso mostrado desde el primer día.

Agradecer a mis padres y a mi hermana el apoyo incondicional que me han dado en todo momento durante mi trayecto por la universidad.

Por último, agradecer a todos esos compañeros que al final han resultado ser amigos que juntos hayamos logrado llegar al final del camino.

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN

Es evidente que hoy en día estamos en medio de una revolución tecnológica, y en especial en estos últimos años se está viendo como se está potenciando el uso de los aviones no tripulados. Tanto es así, que el gobierno se está viendo obligado a regularizar toda una serie de normativas relativas al pilotaje de estos aparatos. Los primeros prototipos empezaron a crearse después de la 1ª Guerra Mundial y ya se empezaron a usar en la 2ª Guerra Mundial, y es a medida del paso de los años donde se comienzan a fabricar prototipos cada vez más pequeños y más ligeros.

A día de hoy los drones pueden estar equipados con cámaras de alta definición u otros sensores de adquisición de datos e incluso pequeñas placas que realizan el procesado de dichos datos en tiempo real, y es en este punto donde se centrará el proyecto. Como se ha nombrado anteriormente el primer uso de estos dispositivos era para conflictos bélicos pero con el paso del tiempo y la mejor autonomía de ellos se ha conseguido expandir entre un mercado más comercial.

Este proyecto pretende explorar la posibilidad de modelar el sistema procesador como una red de colas y utilizar técnicas de simulación para validar la configuración del sistema previamente a la realización de una misión, permitiendo determinar cuál es la mayor tasa de adquisición a la que se puede trabajar con unos determinados recursos sin formar una cola que produzca la saturación en la red. Las imágenes adquiridas actúan como clientes que solicitan una secuencia de servicios, siendo éstos los diferentes algoritmos de procesado que se ejecutan en los núcleos de procesamiento de la placa.

Para proceder a una simulación que permita obtener unos resultados más representativos del caso real, es necesario ajustar al máximo los parámetros del sistema. Mediante un ajuste de la distribución de probabilidad del tiempo de servicio, se van a obtener valores que servirán para decidir qué distribución emplea cada algoritmo. Por otro lado, el tiempo entre llegadas se puede modelar con una distribución de probabilidad determinista, que depende únicamente de la frecuencia de adquisición del sensor. Como veremos a lo largo del proyecto, esto va a permitir modelar de una forma más agresiva y acercarnos a tasas de llegada más altas.

Para llevar a cabo este proceso se han utilizado herramientas que han permitido la simulación y el análisis en un primer lugar de cada algoritmo de forma aislada y luego de la red completa. Mientras que los sistemas exponenciales pueden resolverse de forma analítica, para resolver sistemas más complejos o con distribuciones arbitrarias distintas de la exponencial es necesario recurrir a técnicas de simulación. Las herramientas utilizadas son OMNeT++ y R.

1.1 Objetivos del proyecto

Como se ha expuesto en el apartado anterior, el objetivo principal del proyecto es explorar la posibilidad de utilizar técnicas de simulación de redes de colas para la validación de sistemas de procesado en tiempo real. Aunque la idea nace de la necesidad de realizar procesado a bordo de aviones no tripulados, el mecanismo puede ser extrapolado a otros ámbitos distintos.

Para abordar este objetivo se han planteado los siguientes retos:

- Encontrar un entorno de trabajo que permita simular redes de colas especificando las distribuciones de probabilidad de los tiempos de llegada y del tiempo de ejecución, así como el número de servidores por nodo u otros parámetros característicos de teoría de colas.

- Evaluar el comportamiento de un conjunto de algoritmos, tanto de forma analítica como mediante simulación, utilizando un modelo de colas exponencial.

- Ajustar las funciones de distribución de probabilidad de los tiempos de ejecución de los algoritmos colas.

- Evaluar mediante simulación el comportamiento de los algoritmos anteriores utilizando las distribuciones de probabilidad correspondientes.

- Proponer un mecanismo para determinar la tasa máxima de llegadas que puede soportar un algoritmo para una configuración (número de núcleos de procesamiento) determinada.

- Evaluar mediante simulación de uno o más casos de usos el comportamiento de una red de colas (secuencia de algoritmos para llevar a cabo una misión determinada).

1.2 Contenido de la memoria

Antes de empezar con el contenido, es importante ver como se han dividido los capítulos de la memoria. Contiene 5 puntos generales, los cuales se pasan a explicar a continuación:

- Capítulo 1: Este primer punto trata de introducir al lector lo que se ha llevado a cabo en este proyecto. La importancia de utilizar la teoría de colas para saber el grado de servicio de un sistema o ajustar distribuciones para conseguir un mejor servicio, son aspectos que se presentan en el capítulo 1. Además, se definen los objetivos a alcanzar con la realización de este proyecto.

- Capítulo 2: El principal objetivo de este capítulo, es mostrar un análisis del sistema utilizando un modelo de colas exponencial. Para ello, en primer lugar, se introducen conceptos generales de la teoría de colas, para mostrar una base teórica de temas que se tratan durante la memoria. Se introducen las herramientas utilizadas, para tener una percepción más clara de cómo se ha ido trabajando y se presentan de un modo más general los algoritmos que se han utilizado, para tener conciencia en que campos se va a poder aplicar este estudio. Finalmente, se va a mostrar una comparativa de valores simulados frente a valores reales, todo ello utilizando un modelo de colas exponencial, con el objetivo de ver que efectivamente las simulaciones son correctas.

- Capítulo 3: Se muestran los métodos utilizados para el ajuste de la distribución de probabilidad del tiempo de servicio. Se obtienen resultados necesarios para la realización del siguiente capítulo. Se puede ver de qué manera y cuáles son las distribuciones elegidas para cada algoritmo.

- Capítulo 4: En este capítulo, se ejecutan y se analizan dos casos de uso con estos algoritmos. El primero de ellos, será un conjunto de colas con un sistema multiprocesador que se utiliza para la detección de incendios. Será necesario establecer un mecanismo para determinar la tasa máxima de llegadas para cada algoritmo. Se conseguirá de esta manera unos resultados finales que mediante la comparación de los mismos algoritmos pero con un sistema de colas exponencial, se podrá demostrar que efectivamente trabajar con distribuciones arbitrarias permite realizar una planificación más agresiva. Del mismo modo, se analizará el caso de uso de Jellyfish utilizado para la detección de medusas, en este caso sin embargo, es el propio algoritmo el que es capaz de procesar los datos y obtener resultados óptimos, realizando a posterior una comparación utilizando un modelo de colas arbitrario frente a uno exponencial.

- Capítulo 5: Básicamente se cierra el proyecto a través de unas conclusiones, analizando los resultados finales obtenidos en los diferentes capítulos de la memoria.

Además de estos cinco capítulos principales, se incluye en la memoria el apartado de anexos, donde se muestran gráficos y datos adicionales del resto de algoritmos utilizados.

Para concluir, se incluye un apartado con toda la bibliografía utilizada para el desarrollo de este proyecto.

CAPÍTULO 2. ANÁLISIS DEL SISTEMA UTILIZANDO EL MODELO DE COLAS EXPONENCIAL

El objetivo de este primer capítulo, no es otro que presentar de forma teórica conceptos generales que van a servir para introducirnos en la teoría de colas (ver [1] y [3]). Además, se presentarán las herramientas utilizadas para la producción del entorno de trabajo y a continuación se mostrarán los algoritmos con los que se ha trabajado. Una vez presentados estos conceptos teóricos, se explicará y se mostrará de forma gráfica una comparativa de una cola M/M/k entre valores teóricos y valores generados a partir de las simulaciones.

2.1 Conceptos generales de teoría de colas

2.1.1 Introducción

Existe un fenómeno habitual en la vida real como son las formaciones de colas o llamado de otro modo, las líneas de espera. Decimos que esto se produce cuando los clientes llegan a un lugar demandando un servicio. Dicho servicio necesita un servidor que le atienda, el cual, tiene una cierta capacidad de atención. La cola se forma cuando existe una demanda de un servicio que es superior a la capacidad que tiene el servidor para dar ese servicio. Cuando un servidor no está disponible inmediatamente y el cliente se mantiene en espera, es entonces cuando se forma la línea de espera.

Una cola al fin y al cabo es una línea de espera y la teoría de colas es una colección de modelos matemáticos que describen sistemas de colas. Mediante los modelos podemos encontrar un buen compromiso entre costes del sistema y los tiempos promedios de la línea de espera para un sistema dado.

2.1.2 Teoría de colas

El estudio matemático de las colas o líneas de espera dentro de un sistema se denomina teoría de colas. Esta teoría estudia factores como el número de clientes que insertan en el sistema, el tiempo de espera medio en las colas o la capacidad de trabajo del sistema sin que llegue a colapsarse.

Mediante la teoría de colas podemos modelar sistemas en los que varios clientes, que en este proyecto son las imágenes que entran al sistema, demandan cierto servicio o prestación, que son atendidas por un mismo servidor y, por lo tanto, pueden generarse esperas desde que un cliente llega al sistema y el servidor atiende sus demandas. En este sentido, como se va a poder ver, la teoría de colas es muy útil para modelar procesos tales como la llegada de datos a una cola.

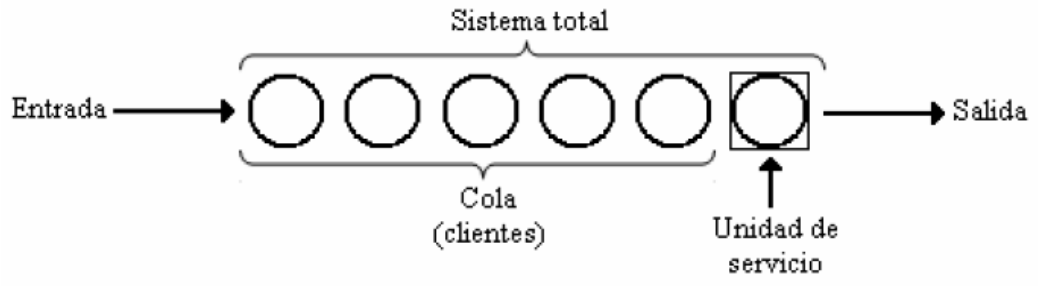

Figura 2.1 Modelo de un sistema de colas

2.1.3 Formación de colas

Una cola viene formada por el desequilibrio temporal que existe entre la demanda del servicio y la capacidad del sistema para suministrarlo. Las imágenes que se introducen al sistema pueden esperar en cola debido a que los medios existentes sean inadecuados para satisfacer la demanda del servicio; en este caso, la cola tiende a ser cada vez más larga a medida que pasa el tiempo. Los clientes pueden esperar temporalmente, porque los clientes llegados anteriormente están siendo atendidos, pero también es posible que en algunos casos se abandone el sistema si el servidor se ve abordado y los clientes se cansan de esperar.

2.1.4 Características de los sistemas de colas

Existen seis características básicas que se deben utilizar para describir adecuadamente un sistema de colas. Estas características son las siguientes:

1. Patrón de llegada de los clientes
2. Patrón de servicio de los servidores
3. Disciplina de cola
4. Capacidad del sistema
5. Número de canales del servicio

2.1.4.1 Patrón de llegada de los clientes

Ante una situación de cola habitual, la llegada es estocástica, es decir la llegada depende de una cierta variable aleatoria, para ello es necesario conocer la distribución probabilística entre dos llegadas sucesivas. Al mismo tiempo, se tendría que valorar si los clientes llegan de una manera

independiente o simultáneamente. De ser el segundo caso, es decir si los clientes llegan en lotes, hay que definir la distribución probabilística de éstos.

También es posible que ante una cola demasiado larga o tras mucho esperar, los clientes “impacientes” decidan abandonar.

En último lugar, es posible que el patrón de llegada varíe durante el tiempo. Le llamamos estacionario, si el éste patrón de llegada se mantiene constante y no estacionario si va variando a lo largo del tiempo.

2.1.4.2 Patrón de servicio de los servidores

Los servidores pueden tener un tiempo de servicio variable, en el cual para definirlo hay que asociarle una función de probabilidad. Éstos también pueden atender a los clientes de forma individual o por lotes.

El tiempo de servicio de un servidor puede variar en función del número de clientes en cola, trabajando así de una manera más rápida o más lenta. En este proyecto, el tiempo de servicio variará dependiendo del algoritmo, veremos que los servidores trabajan con tiempos de servicio diferentes.

Del mismo modo que el patrón de llegadas, el patrón de servicio puede mantenerse constante (estacionario) o variar en función del tiempo (no estacionario).

2.1.4.3 Disciplina de cola

La disciplina de cola se refiere al orden en el que se seleccionan sus miembros para ser atendidos. Cuando pensamos en colas se admite que la disciplina de cola normal es la FIFO (atender primero al que llegó primero). No obstante, también es habitual ver la disciplina LIFO (atender primero al último). Se va a ver que en este proyecto la disciplina usada es FIFO y será atendida la primera imagen en llegar.

En otros casos de simulación de colas, se puede encontrar reglas de secuencia con prioridades, según la duración del servicio o el tipo de cliente.

2.1.4.4 Capacidad del sistema

Existen sistemas de colas con limitaciones respecto al número de clientes que pueden atender en cola. Estos casos son las denominadas situaciones de colas finitas. Esta limitación se puede considerar como una simplificación en la modelización producida por la “impaciencia” de los clientes.

En este proyecto se utilizarán colas infinitas para realizar el estudio, ya que estamos interesados en obtener las métricas relacionas con el número de clientes en cola o el tiempo medio de espera en cola. En una situación real hay que decidir entre implementar una cola pequeña o permitir que se pierdan imágenes si no pueden ser atendidas en el momento de su llegada, suponiendo en ambos casos que el sistema se ha planificado de forma que esto ocurre con poca frecuencia. La decisión dependerá básicamente de lo crítico que sea perder una imagen.

2.1.4.5 Número de canales del servicio

En general, es preferible utilizar sistemas multiservicios con una única cola de espera para todos, que una línea de espera por servidor. Cuando se habla de canales de servicios paralelos, se habla generalmente de una cola que alimenta a varios servidores mientras que el caso de colas independientes existe una cola por cada servidor.

En la siguiente imagen se presenta las dos variantes. El primero con una sola cola de espera, mientras que el segundo tiene una línea de espera por canal.

Figura 2.2. Ejemplos de sistemas de colas de espera multicanal

En este proyecto se utilizará una única cola para alimentar a cada algoritmo o servicio, del cual puede haber diferentes instancias (servidores) en los distintos núcleos de procesamiento.

2.1.5 Nomenclaturas básicas

Clasificamos los diferentes modelos de colas según la notación estándar implantada por Kendall. Consiste en una serie símbolos separas entre sí por barras.

A/B/K/C/N

Dónde:

A: Distribución del tiempo entre llegadas (D: Determinista, M: Markov)

B: Distribución del tiempo de servicio (G: General, M: Markov)

K: Numero de servidores en el sistema

C: Capacidad total del sistema (cola +servidores)

N: Número de usuarios de la población.

Disciplina de colas (FIFO, LIFO, Prioridades, etc.)

En cuanto a terminología, el estándar en la teoría de colas es el siguiente:

- N(t) = Número de clientes en el sistema de colas en el tiempo t
- Pn (t) = Probabilidad de que exactamente n clientes estén en el sistema en el tiempo t.
- k = Número de servidores
- λn = tasa media de llegadas de nuevos clientes cuando hay n clientes en el sistema.
- μn = tasa media de servicio en todo el sistema cuando hay n clientes en el sistema (tasa combinada de todos los servidores).
- 1/λ = tiempo esperado entre llegadas
- 1/μ = tiempo esperado de servicio
- ρ = λ/μ Factor de utilización
- W= Tiempo medio que un paquete permanece en el sistema
- Wq= Tiempo medio de espera en cola
- L = número de clientes en la cola
- Estado del sistema = Número de clientes en el sistema
- Longitud de la cola = Número de clientes que esperan servicio

En la siguiente tabla se representa un resumen de los símbolos más utilizados en la teoría de colas:

Tabla 2.1. Símbolos más utilizados en teorías de colas

Característica	Símbolo	Descripción
• Distribución de tiempos de llegada • Distribución de tiempos de servicio	• M • D • Ek • Hk • PH • G	• Exponencial • Determinista • Erlang tipo-k (k=1,2,3…) • Mezcla de exponenciales • Tipo fase • General
• Número de servidores	1,2,…, ∞	
• Disciplina de cola	• FIFO • LIFO • RSS • PR • GD	• Servir al que primero que llegue • Servir al último que llegue • Selección aleatoria de servicio • Prioridad • Disciplina general

2.1.6 Distribución exponencial y proceso de Poisson

La gran parte de los modelos de colas estocásticas asumen que el tiempo entre las diferentes llegadas de clientes sigue una función exponencial. O dicho de otra manera, que el ritmo de llegadas de los clientes o paquetes sigue una distribución de Poisson.

Podemos decir que también es habitual admitir que el ritmo en el que los servidores atienden a los clientes tienen una distribución de Poisson y la duración de la atención es una distribución exponencial.

Como bien veremos en este proyecto, los análisis realizados en la primera parte de éste, se realizan siguiendo una función exponencial, para más adelante hacer una comparación con la función que consiga modelar de una manera más efectiva los tiempos de espera de los algoritmos en cuestión.

2.1.7 Distribuciones estadísticas

Aun siendo la función exponencial la más típica en un modelo de colas, no todas las llegadas ni todos los servicios se pueden simular mediante ésta. Existen otras distribuciones que se ajustan mejor a otros procesos reales y es función del modelador elegir la función que mejor se ajusta a la realidad para así poder obtener resultados razonables.

A continuación vemos las principales distribuciones estadísticas de tipo discreto.

2.1.7.1 Distribuciones estadísticas de tipo discreto

Estas distribuciones de tipo discreto cogen valores de un conjunto finito de posibilidades. Nos permiten representar el número de clientes dentro de un intervalo de tiempo.

- Si las ocurrencias son un conjunto finito y uniforme de valores, se conoce como variable **Uniforme Discreta**.

- La probabilidad de obtener k sucesos A con probabilidad p, a partir de n intentos, se le denomina distribución **Binomial**.

- Si la probabilidad de ocurrencia es diferente, donde la variable solo puede tomar dos valores y con una cierta probabilidad p para el primero de los dos, se conoce como distribución de **Bernouilli**.

- Encontramos también la distribución **Geométrica**, que básicamente es la encargada de representar la probabilidad de obtener la primera ocurrencia A en el lanzamiento n.

- La ya conocida distribución de **Poisson** en la que se representan ocurrencias para un gran conjunto e independientes. Es la más utilizada.

2.1.7.2 Distribuciones estadísticas de tipo continuo

Son las adecuadas para representar intervalos de tiempo entre eventos consecutivos.

- La **exponencial** (o negativa exponencial) es la complementaria de la distribución de Poisson. Se suele utilizar para representar el tiempo que transcurre entre dos ocurrencias consecutivas de eventos independientes.

- La **continua uniforme** toma valores equiprobables en un determinado rango [a,b].

- La **Erlang[k,β]** es una distribución que es la suma de k exponenciales de media β/k.

  La distribución Erlang es parte de una distribución más amplia que son las distribuciones **gamma**, que vienen definidas por dos parámetros α y β.

- La distribución **Weibull**, que se encarga de modelar la distribución de fallos en el sistema, cuando la tasa de fallos es proporcional a una potencia del tiempo.

- Una variable puede ser modelada como **log-normal** si puede ser considerada como un producto multiplicativo de muchos pequeños factores independientes. Un ejemplo típico es un retorno a largo plazo de una inversión: puede considerarse como un producto de muchos retornos diarios. La distribución log-normal tiende a la función densidad de probabilidad.

La selección de la distribución estadística que más se ajuste a la realidad debe ser realizada mediante procedimientos estadísticos estándar de captura de datos y validaciones de hipótesis.

2.1.8 Conclusión

Las anteriores características, sirven por lo general para definir cualquier proceso. Lógicamente, se pueden encontrar diferentes problemas, por eso antes de comenzar cualquier análisis matemático se debe describir adecuadamente el proceso en función de las características descritas en los puntos anteriores.

Una mala elección del modelo lleva a unos resultados erróneos, y el no analizar correctamente el sistema nos puede llevar a pensar que no es posible de modelar.

2.2 Entorno de trabajo

En este subcapítulo se definen las dos herramientas fundamentales que se han utilizado para llevar a cabo las simulaciones y el análisis de los modelos de colas empleados en este proyecto.

2.2.1 Herramienta de simulación RStudio

RStudio es un software libre y con entorno de desarrollo integrado (IDE) para el lenguaje de programación R, para computación estadística y gráficos. En definitiva R es un conjunto integrado de programas para manipulación de datos, cálculo y gráficos (ver [11]).

Incluye una consola, editor de sintaxis que apoya la ejecución de código, así como herramientas para el trazado, la depuración y la gestión del espacio de trabajo.

2.2.1.1 Entorno de programación RStudio

RStudio tiene la misión de proporcionar el código abierto más utilizado y software profesional lista para la empresa para el entorno informático estadístico R. Estas herramientas más la causa de equipar a todos, independientemente de los medios, para participar en una economía mundial que recompensa cada vez alfabetización datos.

Ofrece código abierto y herraminentas adaptadas a su empresa para el medio ambiente de computación. Al ser un entorno de desarrollo integrado (IDE), hace que sea fácil para cualqueir persona analizar los datos con R. También ofrece muchos paquetes R y una plataforma para el intercambio de aplicaciones interactivas e informes reproducibles con otros.

2.2.1.2 ¿Por qué utilizar R?

Estadística se puede hacer con miles de paquetes estadísitcos, incluso con hojas de cálculo e incluso con lápiz y papel, pero... ¿Que tiene R que lo hace especial?

- Es libre. Se distribuye bajo licencia GNU, lo cual significa que se puede utilizar y mejorar.
- Es multiplataforma, hay versiones para Linux, Windows, Mac, iPhone...
- Se puede analizar con R cualquier tipo de datos.
- Es muy potente.
- Su capacidad gráfica difícilmente es superada por ningún otro paquete estadístico.
- Es compatible con ‘todos’ los formatos de datos (.csv, .xls, .sav, .sas…).
- Es ampiable.
- Hay miles de técnicas estadísticas implementadas, cada día más.

2.2.2 Herramienta de simulación OMNeT++

En este apartado vamos a describir una de las herramientas de software propuestas para la realización de este proyecto orientada al dimensionamiento de una red de comunicaciones, mediante técnicas de modelado por simulación. Se trata de una aplicación dotada de un amigable interfaz de usuario, que

facilita tanto la ejecución de los pasos como el posterior análisis de los resultados obtenidos (ver [6] y [7]).

La simulación de sistemas reales mediante una herramienta software comprende técnicas para su estudio cuantitativo, que consiste en primer lugar, en definir el modelo matemático del sistema real para representar numéricamente la evolución del modelo durante un cierto periodo de tiempo, estimar características de interés del sistema a través de los datos recogidos en el paso anterior. Estas técnicas tienen como ventaja frente a otros métodos de estudio, que permiten abordar sistemas reales cuando ésta es imposible o muy costosa.

Para ello, se ha elegido la herramienta de software OMNET++ que pasamos a ver más detalladamente en el siguiente punto.

2.2.2.1 Introducción a la herramienta

La historia del desarrollo de este simulador (Objective Modular Network Testbed in C++) OMNeT++ empezó en 1992 dónde su autor, András Vargas empezó a desarrollarlo a partir del simulador Omnet escrito en Pascal a su vez, por el profesor Dr.Gyrgy Pongor en la Universidad Técnica de Budapest.

OMNeT++ es una versión libre para fines académicos pero también existe OMNEST que es la versión comercial desarrollada por Omnest Global Inc.

OMNeT++ es una herramienta de modelado y simulación pública, basado en componentes modulares y con un ambiente de simulación de arquitectura abierta y con fuerte soporte de GUI.

Es un simulador de ambiente discreto y su área de aplicación primaria es la simulación de redes de comunicación, debido a su arquitectura genérica y flexible ha sido utilizada exitosamente en redes basadas en colas de espera y arquitectura de hardware. Además, múltiples modelos de simulación de fuente abierta han sido publicados, en las simulaciones de Internet (IP, IPv6, MPLS, etc.), movilidad, simulaciones ad-hoc entre otras áreas. Puede ser utilizado en plataformas UNIX y Windows.

OMNeT++ utiliza el lenguaje NED, que está basado en C++, para crear las topologías de red. De este modo, cualquier modelo de OMNeT++ está creado por módulos que pueden contener estructuras complejas de datos y sus propios parámetros.

2.2.2.2 Funcionamiento de OMNeT++

OMNeT++ proporciona las herramientas básicas para realizar simulaciones, pero por sí mismo no proporciona ningún componente específico para la simulación de redes de computadoras, simulaciones de colas o cualquier otra

área. No obstante, estas áreas de aplicación son proporcionadas por varios modelos de simulación.

Lo que OMNeT ++ proporciona en sí es una librería clase C++ que permite la creación de componentes de simulación como módulos simples o canales, se proporciona la infraestructura para reunir las simulaciones de estos componentes y configurarlos en el lenguaje NED o en archivos tipo ini.

El simulador trata de facilitar el trabajo al usuario, para ello cuenta con una herramienta gráfica para el código NED, llamado GNED con el cual generamos el código para configurar y simular las redes.

2.2.2.3 Simulación con OMNeT++

Como bien hemos definido antes, OMNeT++ nos brinda una arquitectura modular. Los modelos son ensamblados por componentes reutilizables: módulos, que bien configurados son reutilizables y pueden ser combinados de distintas maneras. Los módulos pueden conectarse con otros a través de puertos, y combinarlos entre sí para formar módulos compuestos. Las conexiones son creadas dentro de un nivel básico de jerarquía: un sub-módulo puede conectarse con otro igual, o con aquel que contenga en sí un módulo compuesto. Tanto los componentes como la topología se relacionan directamente con los archivos NED. A continuación vemos unos patrones de cómo utilizar OMNeT++:

- Definir la estructura de la red, mediante el lenguaje NED. Los archivos NED se pueden editar en cualquier editor de texto o en GNED, que es el editor gráfico de OMNeT++.

- Se proporciona un archivo omnetpp.ini dónde deberemos configurar la simulación y añadir los parámetros necesarios del modelo creado.

- Creamos el programa de simulación y se corre. En este momento se crea un vínculo entre el código de OMNeT++y la interfaz de usuario.

- Los resultados de la simulación se escriben en un vector y un archivo escalar de salida. Estos archivos nos muestran los resultados finales de la simulación en el fichero ANF.

Por último, en necesario remarcar que existen varias interfaces de usuario. Como veremos más adelante, el simulador consta de varias interfaces de usuario:

- La interfaz de usuario gráfico: En la cual sólo nos deja hacer una simulación en una misma ejecución (en nuestro caso tomando únicamente un valor de lambda).

- Interfaz por línea de comandos: En la que es útil para realizar simulaciones por lotes. En la que si añadimos el *wildcare* * reflejará los valores obtenidos en más de una simulación de una misma ejecución.

2.3 Descripción de los algoritmos

En esta sección se va a definir y explicar en qué consisten los algoritmos usados en este proyecto. Para la evaluación, hemos seleccionado un conjunto de cinco algoritmos de procesamiento de datos que pueden ser de interés para una misión de detección de objetos: Hotspot, Jellyfish, Georef, Fusion y Quality.

En este proyecto se tratará por un lado con una red de colas formada por los algorimos Quality, Hotspot, Georef y Fusion y por otro lado el caso de uso del Jellyfish, creado principalmente para la detección de bancos de medusas (ver [9] y [10]).

2.3.1 Hotspots

Hotspots es un algoritmo de segmentación simple para la detección de puntos calientes en imágenes térmicas. Los pixeles sobre una temperatura umbral dada se agrupan en hotspots. Para cada hotspot, el algoritmo anota información acerca de su centro de masa, caja delimitadora, número de pixeles y temperatura. El resultado es un archivo de texto con la información de los hotspots detectados. Las cajas delimitadoras también están marcadas como rectángulos verdes en la imagen de salida.

2.3.2 Jellyfish

Jellyfish también es un algoritmo de segmentación pero más complejo que el Hotspots. Sirve para detectar grandes bancos de medusas en costas utilizando imágenes aéreas de alta resolución. Para cada imagen, se utiliza una fase inicial para intensificar las diferencias de píxeles. Seguidamente, una segmentación de color para encontrar las regiones de interés de la imagen capturada.

El agrupamiento se realiza aplicando en primer lugar un proceso morfológico de apertura y luego utilizando un etiquetado de componente conectado. Se calcula las diferentes propiedades de los grupos, como el área, el número de píxeles y el cuadro delimitador. Al igual que en el algoritmo anterior, las cajas de contorno también se marcan en la imagen de salida.

2.3.3 Georef

El algoritmo Georef se caracteriza por determinar las coordenadas geográficas de un píxel dado, usando georreferenciación directa. Los datos de entrada son la posición y altitud de la plataforma aérea en el momento de capturar la imagen, los parámetros externos e interno de la cámara y la elevación del

terreno. Este algoritmo no realiza tareas de procesamiento de imágenes, sino que realiza el cálculo del punto de localización.

2.3.4 Fusion

El algoritmo Fusion crea una nueva imagen superpuesta de información térmica sobre la imagen visual. Esta imagen fusionada, debería ayudar a los bomberos a localizar el punto caliente en el campo para mejorar su situación. Tanto las imágenes térmicas como las visuales se rectifican primero y se escalan adecuadamente. La información que nos ha ofrecido Georef se utiliza entonces para calcular la posición de la imagen térmica sobre la visual. Finalmente, se escribe la imagen TIFF georreferenciada resultante. Los datos de entrada incluyen las imágenes, la posición y la actitud de la plataforma aérea, los parámetros externos e internos de ambas cámaras y la elevación del terreno.

2.3.5 Quality

Quality, como su propio nombre marca, mide la calidad de una imagen basada en las métricas de desenfoque y entropía. Una imagen borrosa es aquella cuyos bordes y formas no están claramente definidos. La métrica de desenfoque calcula un grado de nitidez en función del número de píxeles de borde detectados por la función Canny. La imagen se borra primero para reducir la cantidad de ruido presente en la imagen, eliminando muchos bordes falsos. La entropía de la imagen se utiliza para detectar sobre-exposición o sub-exposición. Si el brillo medio de la imagen es demasiado alto o demasiado bajo, se convierte en una imagen homogénea (baja entropía). El histograma HSV normalizado se utiliza para obtener la función de densidad de probabilidad de los colores en la imagen.

2.3.6 Tabla de tiempos de ejecución

Esta sección trata de mostrar el resultado de ejecutar 100 veces cada uno de los algoritmos en la placa ODROID-XU3 (ver [12]). La siguiente tabla muestra la media, la desviación estándar y el coeficiente de variación, que se define como la desviación estándar dividida por la media.

Se observa en la tabla que la varianza del tiempo de ejecución es pequeña, lo cual indica que se alejan bastante de la exponencial (cuyo coeficiente de variación es de 1).

Tabla 2.2 Tiempos de ejecución de la placa ODROID-XU3

Algoritmo	Media	Desviación estandar	Coeficiente de variación
Fusion	1.25 s	0.008826	0.007036
Georef	0.003473 s	0.000104	0.029894
Hotspot	0.035666 s	0.002277	0.063836
Jellyfish	7.794005 s	0.116758	0.014981
Quality	0.5548 s	0.012808	0.023086

2.4 Gráficas M/M/K. Valor teórico vs simulación

El primer paso, ha sido la puesta marcha del escenario. Después de la instalación de OMNeT++ y R se han ido generando las primeras líneas de código y las posteriores simulaciones. Se ha creído oportuno en primer lugar, hacer un análisis de los algoritmos siguiendo un modelo de cola M/M/k es decir, un modelo de cola con k servidores (en este primer caso 1 servidor), los cuales sirven con una razón exponencial μ y al mismo tiempo permite que el tiempo entre llegadas sucesivas tenga una distribución arbitraria. En este primer caso nos encontramos que al ser exponencial se puede resolver de forma analítica, en nuestro caso de una forma más práctica utilizando los ficheros generados en R donde se ha generado el código necesario para obtener el valor teórico de los puntos de interés.

Para ser más conciso, se han extraído los gráficos de manera independiente para ver cómo se comporta la cola cuando trabaja de manera individual. Al mismo tiempo, se ha generado el código en OMNeT++ para evaluar los datos obtenidos y de este modo realizar una comparación con los valores teóricos. Dicho de otro modo, sabremos si las simulaciones realizadas son válidas al compararlo con los valores teóricos generados.

2.4.1 Ficheros NED

Existen varios ficheros a la hora de realizar una simulación en OMNeT++. Como se ha explicado en el capítulo 2, el primero de ellos es la estructura del modelo definido en el lenguaje NED, que básicamente se especifica el modelo de forma visual (ver [2]). Al analizarlos de forma individual en este primer punto se ha utilizado el mismo modelo global para todos ellos, que se muestra a continuación:

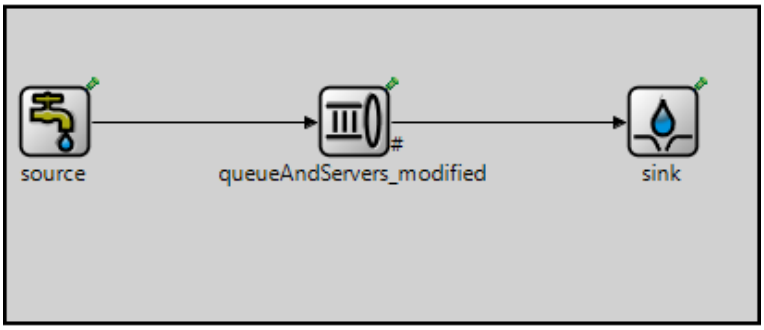

Figura 2.3 Estructura del modelo de colas

En los ejemplos que hemos utilizado para realizar las pruebas, encontramos 3 entidades diferentes: un módulo fuente, un módulo que actúa de cola con un servidor y un módulo sumidero o *sink*. El módulo fuente genera mensajes que representan paquetes de datos. Estos mensajes son enviados a la cola para ser tratados a posterior en la medida de lo posible por un servidor libre. En este caso tratamos con un modelo de colas M/M/1 por lo que únicamente tendremos un servidor. Finalmente el mensaje llega al *sink* donde se procesa y se elimina del sistema, recogiendo valores que nos serán útiles, como el tiempo medio de espera en cola.

Todos estos módulos necesitan tener unos puertos de entrada y salida para enviar y recibir mensajes. Para ello, en el fichero NED se describen estos puertos. Por ejemplo, en el módulo del servidor, vemos las siguientes líneas de código:

```
module QueueAndServers_modified
{
    parameters:
        @group(Queueing);
        @display("i=block/queue;q=queue;bgb=377,305");
        @signal[dropped](type="int");
        @signal[queueLength](type="int");
        @signal[queueingTime](type="simtime_t");
        @signal[numBusyServers](type="int");
        @statistic[dropped](title="drop      event";      record=vector,count;
interpolationmode=none);
        @statistic[queueLength](title="queue                          length";
record=vector,timeavg,max; interpolationmode=sample-hold);
        @statistic[queueingTime](title="queueing     time     at     dequeue";
record=vector,mean,max; unit=s; interpolationmode=none);
        @statistic[numBusyServers](title="busy                        servers";
record=vector,timeavg,max; interpolationmode=sample-hold);

        int capacity = default(-1);     // negative capacity means unlimited
queue
        int numServers = default(1);
        bool fifo = default(true);     // whether the module works as a queue
(fifo=true) or a stack (fifo=false)
        volatile double serviceTime @unit(s);
```

```
    gates:
        input in[];
        output out;

    submodules:
        queue: QueueNoServers {
            gates:
                in[];
                servers[];
        }
        servers[numServers]: ServerIndiv {
            @display("p=294,107");
            gates:
                in;
                out;
        }

        merge: Merge {
            @display("p=234,189");
            gates:
                in[];
                out;
        }
    connections:
        in++ --> queue.in++;
        for i=0..numServers-1 {
            queue.servers++ --> servers[i].in;
            servers[i].out --> merge.in++;
        }
        merge.out --> out;

}
```

El último paso es unir todos los módulos en una red de trabajo. Esto se hace a partir del módulo compuesto que se mostrará a continuación. El cual, se establecen las conexiones entre los módulos obteniendo como resultado un modelo más complejo y aproximado del modelo real a tratar. El aspecto de la sintaxis sería el siguiente:

```
import org.omnetpp.queueing.QueueAndServers_modified;
import org.omnetpp.queueing.Sink;
import org.omnetpp.queueing.Source;


network queue_net
{
    @display("bgb=743,244");
    submodules:
        source: Source {
            @display("p=26,61");
        }
        queueAndServers_modified: QueueAndServers_modified {
            @display("p=172,61");
        }
        sink: Sink {
            @display("p=355,61");
```

```
        }
    connections:
        source.out --> queueAndServers_modified.in++;
        queueAndServers_modified.out --> sink.in++;
```

2.4.2 Fichero .ini

El siguiente fichero es el ini. Este fichero es fundamental para realizar la simulación, ya que en ellos se describen todas las funciones y parámetros a realizar, ya sea el número de simulaciones a ejecutar, la duración de la simulación, el tiempo de servicio con la distribución empleada, etc. El código que se muestra a continuación es el utilizado para Quality:

```
[General]
cmdenv-express-mode = true
output-scalar-file = ${resultdir}/${configname}-${runnumber}.sca
output-vector-file = ${resultdir}/${configname}-${runnumber}.vec
sim-time-limit = 50000s
warmup-period = 25s

[Config queue_net]
network = queue_net
repeat = 1
**.serviceTime = exponential(0.5548s)
**.source.interArrivalTime = exponential(${0.5..2.77 step 0.18}s)
**.maxQueueLength = -1
**.numServers = 1
```

Como podemos ver, el fichero está pensado para un sistema M/M/1. Puede observarse como los parámetros más característicos son el tiempo de servicio y el tiempo entre llegadas de los paquetes que se modelan siguiendo una distribución exponencial. Además, se ha utilizado la simulación por líneas de comandos, ya que nos permite realizar simulaciones por lotes, consiguiendo de este modo hacer una barrido de valores de la tasa de llegadas, reflejando de este modo los valores obtenidos en más de una simulación de una misma ejecución.

También se definen el número de servidores y la longitud de la cola, siendo ésta -1 (infinito). Definimos que los ficheros de salida se guarden cómo tipo escalar y vectorial. Asignamos también un *warmup-period* del sistema, dónde se especifica la duración del periodo inicial de calentamiento. Los resultados que pertenecen a los primeros segundos asignados no se grabarán en los vectores de salida y no serán contados en el cálculo de escalares de salida. Está opción es útil para simulaciones en estado estacionario, ya que se consigue estabilizar el sistema.

Como se explica en el punto 2.2.2.3, existen dos tipos de simulaciones. En estos primeros casos lo que nos interesa es saber el comportamiento del tiempo de espera en relación al tiempo entre llegadas. Por lo tanto, debemos realizar una simulación dónde se vea los diferentes Wq, dependiendo del

1/lambda en la que se encuentre. Dicho de otro modo, se utiliza la interfaz por línea de comandos, en la que nos permite realizar simulaciones por lotes, viendo así los valores obtenidos en más de una simulación de una misma ejecución.

2.4.3 Fichero ANF

Por último y no menos importante es el ANF. Este fichero al fin y al cabo es fundamental, ya que tanto los vectores grabados en la salida como los valores escalares se exponen en este fichero, mostrándolos resultados obtenidos de la simulación y que a posterior, nos servirá para poder crear un fichero CSV y cargarlo mediante R para la posterior comparación con los valores teóricos que nos ofrece R.

2.4.4 Ejecución en R

Una vez obtenidos en OMNeT++ los resultados finales de la simulación agrupados en el fichero ANF, se han utilizado dos scripts generados en R para la comparación prevista de valores teóricos contra valores de simulación. Mediante estos scripts se consigue generar estos gráficos teóricos, gracias a que se hace uso del paquete *queueing* que proporciona R.

Además, los resultados recogidos a través del archivo ANF de OMNeT son cargados por R mediante un fichero CSV.

En este punto se van a mostrar los gráficos de todos los algoritmos simulados, para comprobar que efectivamente el comportamiento es el mismo tanto de forma analítica como mediante la simulación. No obstante, el proyecto se centrará en la explicación de uno de los algoritmos para hacerlo de una manera más llevadera. El algoritmo elegido es el Quality. A continuación vemos los resultados obtenidos de todos los algoritmos:

- Para Quality la gráfica generada es la siguiente:

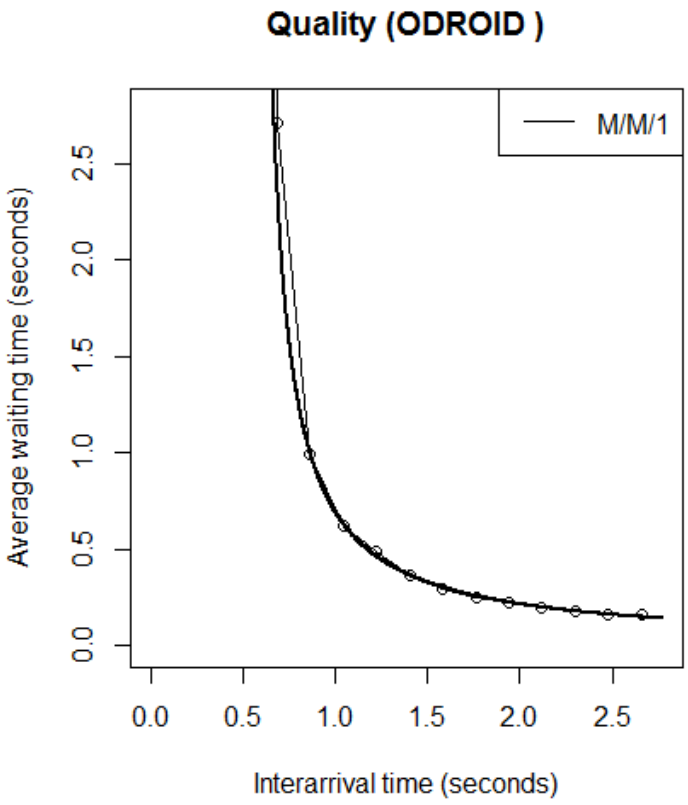

Figura 2.4 Gráfico de valores analíticos frente valores simulados para Quality

La imagen nos muestra cómo se comporta el algoritmo Quality a la llegada de clientes al servidor. Estos clientes son las imágenes procesadas. Vemos de manera gráfica la relación que existe entre el tiempo medio de espera en cola y el tiempo entre llegadas, todo ello expresado en segundos. La línea continua de color negro nos muestra los valores teóricos generados por la herramienta R. En cambio, la línea que une los puntos, son los valores obtenidos después de la simulación en OMNeT++.

Además, a la hora de realizar esta simulación, se ha tenido en cuenta que el número de servidores sea uno. Más adelante, además de ajustar las distribuciones, se ajustará el número de servidores para cada algoritmo en función del tiempo de servicio.

Mostrados los resultados, vemos que los valores que nos ofrece la simulación son correctos y muy similares a los valores analíticos. Por lo tanto, una vez visto que la simulación para un sistema M/M/1 es correcta, en el siguiente capítulo se mostrará el ajuste de las distribuciones de probabilidad del tiempo de servicio.

A continuación se muestran las gráficas obtenidas para los demás algoritmos:

- Gráfica obtenida para el caso de Jellyfish:

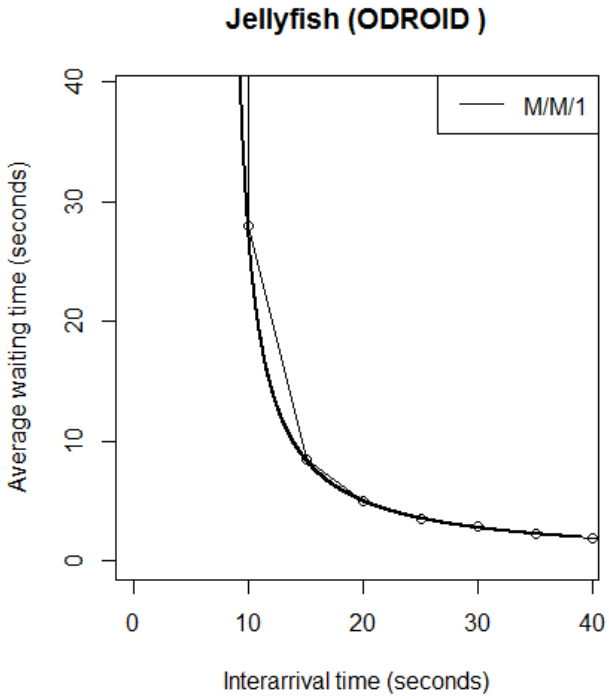

Figura 2.5 Gráfico de valores analíticos frente valores simulados para Jellyfish

- En el caso de HotSpot la gráfica obtenida es la siguiente:

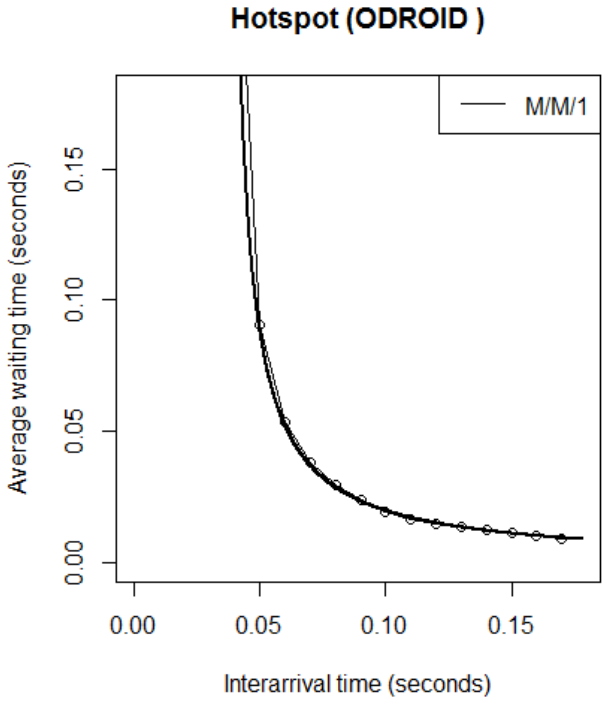

Figura 2.6 Gráfico de valores analíticos frente valores simulados para HotSpot

- Para el algoritmo Georef se obtiene la siguiente gráfica:

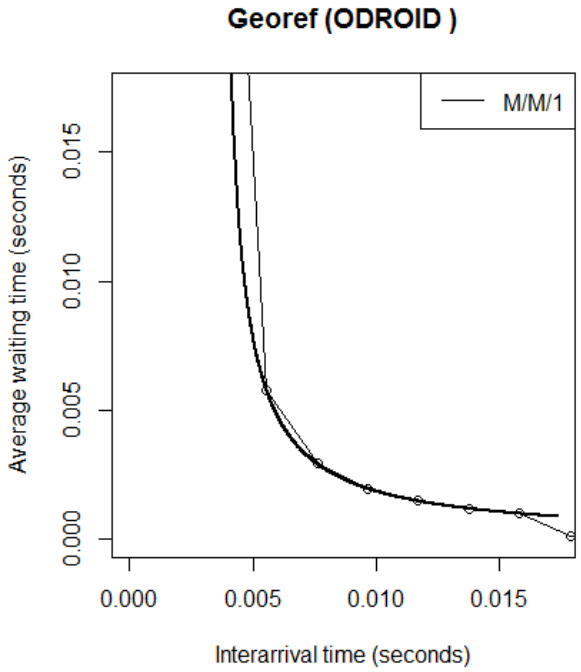

Figura 2.7 Gráfico de valores analíticos frente valores simulados para Georef

- Para el algoritmo Fusion el gráfico obtenido se muestra a continuación:

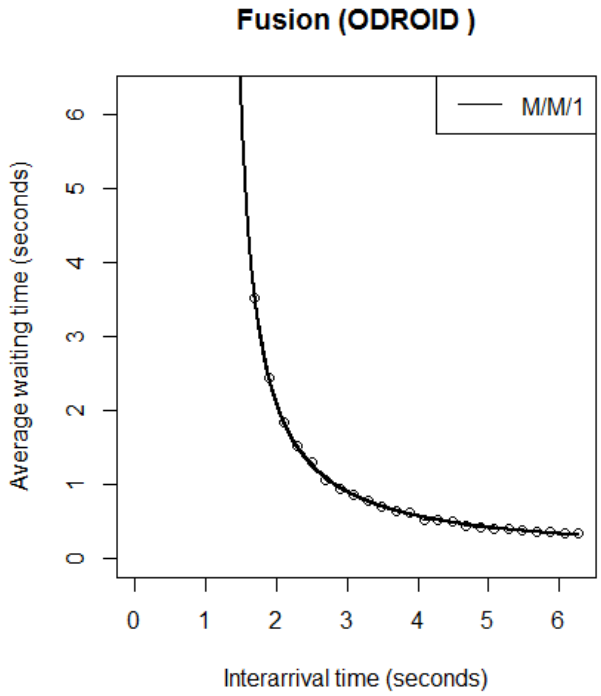

Figura 2.8 Gráfico de valores analíticos frente valores simulados para Fusion

CAPÍTULO 3. Ajuste de la distribución de probabilidad del tiempo de servicio

Como hemos visto en la introducción, el objetivo principal de este proyecto es proceder a técnicas simulación para estimar a qué frecuencia de adquisición podemos trabajar sin formar esperas en cola que produzcan la pérdida de imagenes. El objetivo de este capítulo es ajustar de la mejor manera posible la distribución de probabilidad del tiempo de servicio de cada algoritmo, con el fin de realizar un modelado más realista.

A continuación se sexplicarán la serie de pasos que se han seguido para seleccionar la función de distribución que mejor se aproxime al tiempo de ejecución de cada algoritmo y los parámetros asociados.

En primer lugar, destacar que ajustar distribuciones a los datos es una trabajo muy común en las tareas de estadística y consiste en elegir una distribución de probabilidad que modela la variable aleatoria, así como encontrar estimaciones de parámetros para esa distribución. Para ello se ha utilizado R, que mediante el paquete *fitdistrplus* implementa varios métodos para distribuciones paramétricas y dónde uno de los primeros objetivos en el desarrollo de este paquete fue proporcionar a los usuarios de R un conjunto de funciones dedicadas a ayudar en este proceso global.

La función *fitdistr* estima los parámetros de distribución al maximizar la función likelihood (probabilidad) usando la función óptima y centrándose en la estimación de parámetros desde un punto de vista general (ver [1], [4] y [5]).

3.1 Elección de las distribuciones candidatas

Antes de ajustar una o más distribuciones a un conjunto de datos, se ha tenido que elegir un conjunto predefinido de distribuciones.

Esta elección viene guiada por el conocimiento de los procesos estocásticos o por el conocimiento que se obtiene a partir de las observaciones empíricas, donde básicamente las distribuciones candidatas en un primer lugar han sido las siguientes:

- Exponencial
- Logística
- Weibull
- Normal

- Lognormal
- Gamma
- Beta

Ahora bien, podemos ver como la estadística descriptiva puede ayudar a elegir a los candidatos para describir una distribución entre un conjunto de distribuciones paramétricas. Especialmente el skewness (sesgo) y la kurtosis (curtosis) ligadas al tercer y cuarto momento son útiles para este propósito. Un skewnees diferente de cero nos revela que existe una falta de simetría de la distribución empírica, mientras que el valor de kurtosis analiza el grado de concentración que presentan los valores alrededor de la zona central de la distribución.

Como bien se indicaba en la introducción de este punto, el programa seleccionado para realizar esta parte del proyecto ha sido R y una de las funciones que se han utilizado ha sido *descdist*, que es la encargada de proporcionar estadísticas descriptivas clásicas (mínimo, máximo, mediana, media o desviación estándar), skewness y kurtosis.

Para hacer una descripción más profunda de los pasos que se han seguido, a partir de este momento nos vamos a centrar en el algoritmo Quality, que nos va a servir para ver los resultados que hemos ido obteniendo a lo largo de la realización de este proyecto. La repetición del procedimiento para el resto de algoritmos puede encontrarse en el Anexo de este documento.

A continuación, veremos los parámetros obtenidos después de poner en práctica dicha función *descdist*.

```
> descdist(sdata, boot = 1000)

summary statistics
------
min:  0.01567129   max:  0.9843287
median:  0.532435
mean:  0.5421117
estimated sd:  0.2007222
estimated skewness:  -0.2127347
estimated kurtosis:  3.374672
```

Figura 3.1 Parámetros de la función *descdist* (Quality)

Los valores que vemos en la imagen corresponden a todos esos parámetros comentados anteriormente. Anotar, que el parámetro *sdata* es una tabla con los tiempos de 100 ejecuciones. La función *descdist* que nos proporciona un gráfico skewness-kurtosis como el siguiente:

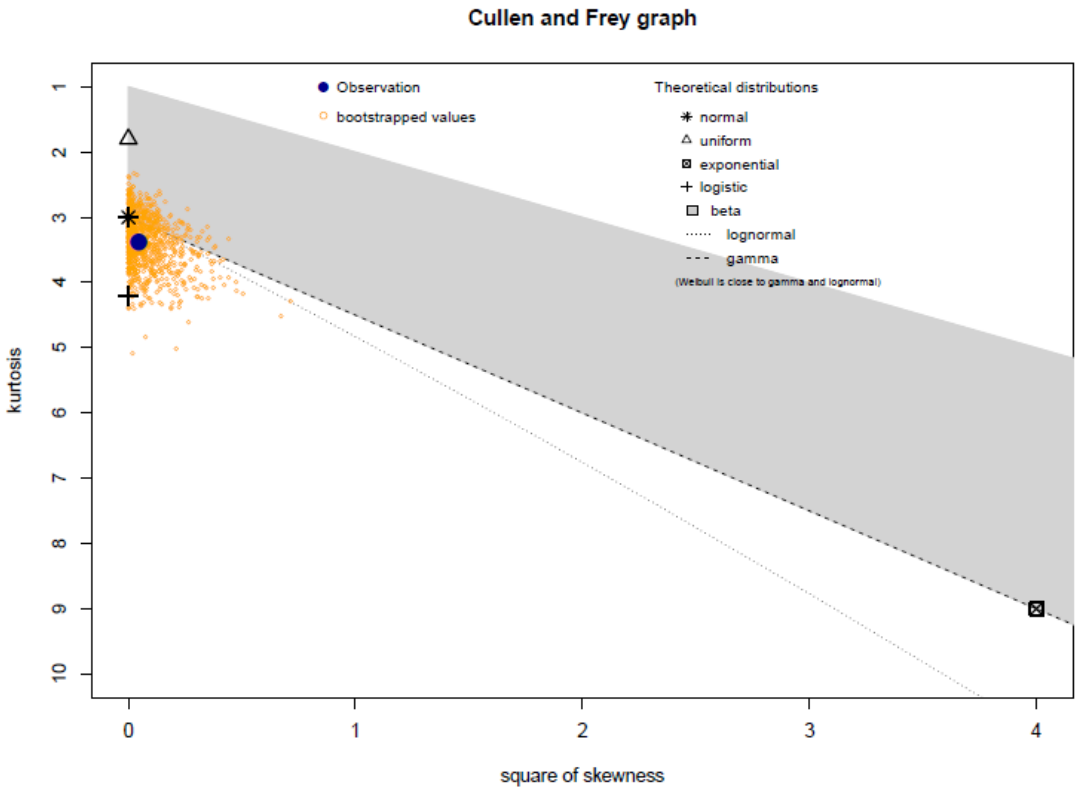

Figura 3.2 Gráfico skewness-kurtosis para una variable continua según lo proporcionado por la función *descdist* para Quality.

En este gráfico, los valores de las distribuciones comunes se muestran con el fin de ayudar a la elección de las distribuciones para el ajuste de datos. Se puede apreciar como para algunas distribuciones (normal, uniforme, logístico o exponencial) solo existe un posible valor para el skewness y la kurtosis. Mientras que para otras distribuciones, se representan áreas de posible valores, que consisten en líneas (distribución gamma y lognormal), o áreas más grandes (distribución beta).

Aun así, debemos saber que el skewnees y kurtosis no son robustas del todo, ya que se calculan con muestras de arranque. Sin embargo son dos parámetros que el usuario debe saber y que como todos los momentos superiores tienen una varianza muy alta. Esto hace que no nos deje claramente marcada la distribución a escoger, y considerando los resultados obtenidos vemos una asimetría negativa y una kurtosis no muy lejos de 3, donde podríamos considerar tres distribuciones comunes como son la normal, lognormal o gamma. Es por ello que para conseguir acercarnos más a una de ellas deberemos proceder al ajuste de las distribuciones.

3.2 Ajuste de distribución

Una vez seleccionado una o más funciones de distribución, el siguiente paso es el ajuste del conjunto de datos usando la función *fitdist* que nos proporciona la librería de R, dónde por defecto, los parámetros de distribución se calculan maximizando la función de probabilidad definida de la siguiente manera:

$$L(\theta) = \prod_{i=1}^{n} f(x_i|\theta) \tag{3.1}$$

Siendo xi las n observaciones de la variable X y f(·|Θ) la función de densidad de la distribución paramétrica.

Básicamente el ajuste de una distribución usando *fitdist* hace que se definan las siguientes funciones d, p y q (densidad, distribución y funciones cuantitativas). Las distribuciones clásicas están definidas de esa manera en la librería, como por ejemplo *dnorm*, *pnorm* y *qnorm* para la distribución normal.

Antes de ver los resultados obtenidos, hace falta destacar que datos nos devuelve la función *fitdist*. De esta manera, los resultados numéricos que ofrece esta función son los siguientes:

- Estimaciones de parámetros
- Los errores estándar estimados
- Valores de probabilidad logarítmica (Loglikehood)
- Criterios de información Akaike y Bayesian (AIC,BIC)
- Matriz de correlación entre estimaciones de parámetros

No obstante, como veremos más adelante nos va ser gran utilidad obtener la lectura de los datos, que no son otros que los valores que nos retorne la instrucción summary, los comentados anteriormente. Para ello, se ha utilizado la siguiente línea de código con sus respectivos resultados:

```
> fwe <- fitdist(data, "weibull")
> summary(fwe)
Fitting of the distribution ' weibull ' by maximum likelihood
Parameters :
        estimate  Std. Error
shape 46.1277130 3.379028468
scale  0.5609475 0.001288344
Loglikelihood:  290.0439   AIC:  -576.0878   BIC:  -570.8774
Correlation matrix:
          shape     scale
shape 1.0000000 0.3334902
```

```
scale 0.3334902 1.0000000

> fln <- fitdist(data, "lnorm")
> summary(fln)
Fitting of the distribution ' lnorm ' by maximum likelihood
Parameters :
          estimate  Std. Error
meanlog -0.5894120 0.002304280
sdlog    0.0230428 0.001615584
Loglikelihood:  294.0875   AIC:  -584.1751   BIC:  -578.9647
Correlation matrix:
              meanlog         sdlog
meanlog  1.000000e+00 -5.290357e-14
sdlog   -5.290357e-14  1.000000e+00

> fex <- fitdist(data, "exp")
> summary(fex)
Fitting of the distribution ' exp ' by maximum likelihood
Parameters :
     estimate Std. Error
rate  1.80245   0.180245
Loglikelihood:  -41.08529   AIC:  84.17059   BIC:  86.77576

> fga <- fitdist(data, "gamma")
> summary(fga)
Fitting of the distribution ' gamma ' by maximum likelihood
Parameters :
      estimate Std. Error
shape 1887.224   266.3788
rate  3401.635   480.1988
Loglikelihood:  294.193   AIC:  -584.386   BIC:  -579.1757
Correlation matrix:
         shape     rate
shape 1.000000 0.999867
rate  0.999867 1.000000
```

Podemos decir, que en este momento tenemos los parámetros de ajuste para cada distribución. No obstante, aún no sabemos cuál de ellas es la que más se ajusta al tiempo de servicio de los servidores de cada algoritmo. Para tener conciencia de ello de una manera más visual, se nos presenta la necesidad de conseguir los gráficos de un objeto de la clase *fitdist*, siendo estas cuatro graficas clásicas. Esto nos va ayudar a hacernos una idea de que distribución tiene que ser la encargada del ajuste de los datos. Aunque como veremos más adelante tampoco nos es totalmente fiable, por lo que tendremos que proceder a un siguiente paso.

Las cuatro gráficas que nos que nos proporcionan son las siguientes:

- Un gráfico de densidad que representa la función de densidad de la distribución ajustada junto con el histograma de la distribución empírica.

- Una gráfica CDF tanto de la distribución empírica como de la distribución ajustada.
- Un gráfico Q-Q representando las cantidades empíricas (eje y) frente a los cantidades teóricas (eje x).

- Un gráfico P-P representando la función de distribución empírica evaluada en cada punto de datos (eje y) en función de la función de distribución ajustada (eje x).

A diferencia de la función general plot, las funciones denscomp, cdfcomp, qqcomp y ppcomp nos van a permitir separar cada una de estas cuatro gráficas para que sea posible la comparación de la distribución empírica y las múltiples distribuciones que se valoren. En este proyecto, vamos a comparar el ajuste de una distribución exponencial, una weibull, lognormal, gamma y beta. Para ello se ha utilizado las siguientes líneas de código:

```
plot.legend <- c("exponential", "weibull", "lognormal", "gamma",
"beta")
denscomp(list(fex, fwe, fln, fga, fbe), legendtext = plot.legend)
qqcomp(list(fex, fwe, fln, fga, fbe), legendtext = plot.legend)
cdfcomp(list(fex, fwe, fln, fga, fbe), legendtext = plot.legend)
ppcomp(list(fex, fwe, fln, fga, fbe), legendtext = plot.legend
```

Observamos que el resultado obtenido es el siguiente:

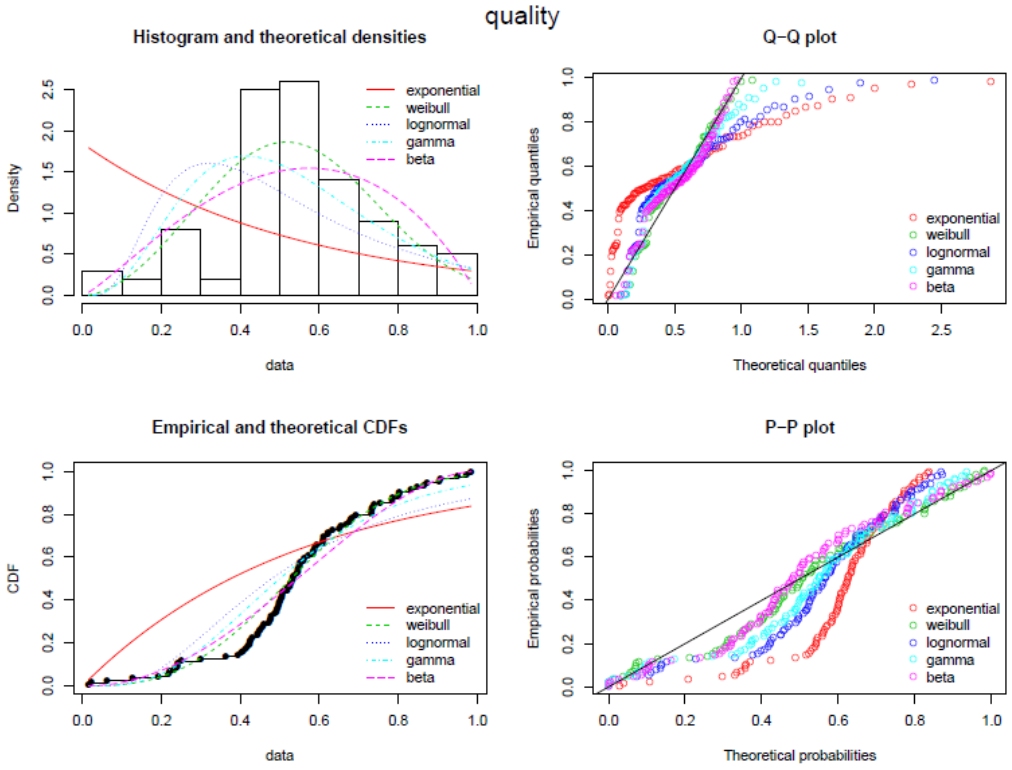

Figura 3.3 Ajuste de bondad para diferentes distribuciones proporcionadas por las funciones denscomp, qqcomp, cdfcomp y ppcomp.

La gráfica de densidad y CDF se pueden considerar cómo gráficas clásicas de bondad de ajuste (goodness-of fit). Las otras dos son complementarias y nos

pueden ofrecer información adicional. Por un lado, el gráfico Q-Q enfatiza la falta ajuste en las colas de distribución mientras que el gráfico P-P enfatiza la falta de ajuste en el centro de distribución. En la figura 3.3, ninguna de las cinco distribuciones describe de una manera concisa el centro de la distribución, pero las distribuciones Gamma y lognormal podrían ser elegidas para la descripción de la cola.

El cálculo de diferentes estadísticas de ajuste de bondad se propone en el paquete *fitdistrplus* con el fin de comparar aún más las distribuciones ajustadas. El objetivo de las estadísticas de ajuste de bondad es medir la distancia entre la distribución paramétrica fija y la distribución empírica; Por ejemplo, la distancia entre la función de distribución acumulada F y la función de distribución empírica Fn. Se consideran tres estadísticas de bondad: Estadísticas de Cramer-von Mises, Kolmogorov-Smirnov and Anderson-Darling statistics, que pueden calcularse utilizando la función *gofstat* definida por D'agostino y Stephens.

La estadística de Anderson-Darling es de especial interés a la hora de seleccionar la distribución de la cola. Se suele usar a menudo para seleccionar la mejor distribución, aun así debe usarse con cautela cuando se comparan ajustes de varias distribuciones.

Es por ello que parece interesante mirar los criterios clásicos basados en la probabilidad logarítmica (AIC,BIC) para salir de dudas en caso de que las haya.

```
> gofstat(list(fwe, fln, fex, fga), fitnames = c("weibull", "lnorm",
"exp", "gamma"))

Goodness-of-fit statistics
                                 weibull     lnorm        exp     gamma
Kolmogorov-Smirnov statistic  0.2170623 0.1164107  0.6270538 0.1170654
Cramer-von Mises statistic    1.6847868 0.2264718  9.9503833 0.2303810
Anderson-Darling statistic   10.0617953 1.9111081 45.3146847 1.9386090

Goodness-of-fit criteria
                                   weibull     lnorm      exp     gamma
Aikake's Information Criterion -584.8667 -660.4972 247.3304 -660.0761
Bayesian Information Criterion -579.6564 -655.2868 249.9355 -654.8658
```

En el ejemplo que se muestra, nos marca que la distribución a elegir es esa que la distancia entre distribuciones sea la mínima. Todas las estadísticas de bondad en base a la distancia están a favor de la distribución lognormal aunque también es cierto que la distribución gamma se encuentra muy cerca. Mientras que los valores marcados por AIC y BIC dan la preferencia a la distribución lognormal y a la distribución gamma respectivamente, Podemos decir que la elección en este caso, basándonos sobre todo en las estadísticas de ajuste de bondad, es la distribución lognormal.

Basándonos en este análisis para la elección de distribución, a continuación presentamos la siguiente la tabla, dónde se especifica la distribución elegida para cada algoritmo y los valores de sus parámetros.

Tabla 3.1. Distribuciones y parámetros obtenidos para todos los algoritmos

	Distribución	**Parámetro 1**	**Parámetro 2**
Hotspot	Gamma	Shape = 185.747	Rate = 5208.017
Jellyfish	Lnorm	Meanlog = 2.05324531	Stdlog =0.01474755
Quality	Gamma	Shape = 1887.224	Rate = 3401.635
Fusion	Lnorm	Meanlog = 0.22662751	Stdlog = 0.00695668
Georef	Lnorm	Meanlog = -5.66306641	Stdlog = 0.02906538

Los algoritmos que se modelan mediante la distribución gamma presentan por un lado el parámetro Shape, que nos indica la "forma" y por lo tanto no tiene unidades y por otro lado nos proporciona el parámetro Rate, que hay que convertirlo a segundos para que los valores sean aceptados a posterior por OMNeT++ y dónde será llamado Scale. Para ello la fórmula empleada es la siguiente:

$$Scale\ (s) = \frac{1}{Rate} \qquad \textbf{(3.2)}$$

Por parte de los algoritmos que necesiten modelar el sistema con una distribución lognormal, el scale se proporciona mediante el parámetro Meanlog y aparece el parámetro de desviación estándar logarítmica, Stdlog, que igual que el shape es adimensional. Al introducir estos valores en OMNeT++, hay que realizar una conversión ya que la distribución lognormal no acepta unidades en sus argumentos y al ser parámetros del tiempo de servicio se necesitan especificar los segundos (ver [8]).

CAPÍTULO 4. Análisis del sistema utilizando el modelo de colas general

Durante este documento hemos ido viendo cómo trabajan los sistemas a través de distribuciones exponenciales que modelan el tiempo de servicio. No obstante, según hemos visto en la Tabla 2.2, no podemos decir que este tipo de distribución sea el más efectivo a la hora de poder sacar conclusiones, ya que no modela adecuadamente el tiempo de ejecución de los algoritmos. Es por ello que en el punto anterior se ha realizado un ajuste de parámetros para obtener la distribución más efectiva del tiempo de servicio de cada sistema.

Una vez realizado este ajuste, se ha realizado un análisis para dos casos de usos, que llamaremos Hotspots y Jellyfish, utilizando una red de colas, donde cada nodo representa un algoritmo en la secuencia de procesado de la imagen. A su vez, cada nodo o algoritmo se modela como una cola general G/G/k. Es decir, un modelo que asume que tanto el tiempo entre llegadas como el tiempo de servicio siguen una distribución arbitraria G (gamma o lognormal, según lo determinado en el capítulo 3) y que se dispone de k servidores (o núcleos) para ejecutar diferentes instancias de dicho algoritmo. En particular, el tiempo entre llegadas sucesivas es determinista para el primer nodo (suponemos que la imágenes se toman con una frecuencia fija y la varianza del tiempo de adquisición es nula).

En el siguiente punto se van a plantear los dos escenarios descritos, con el fin de encontrar la tasa de llegada mínima que permita trabajar al sistema de manera estable y sacando el máximo provecho. A modo de comparación, se presentan también los resultados obtenidos utilizando el modelo exponencial M/M/k.

4.1 Escenario 1 – Caso de uso HotSpot

Antes de profundizar en los resultados obtenidos, vamos a ver de qué manera se ha focalizado el sistema a la hora de realizar las simulaciones. El sistema está compuesto por cuatro de los algoritmos explicados en puntos anteriores (Quality, HotSpot, Georef y Fusion) de los cuales, de forma lógica, cada uno trabaja con un tiempo de servicio diferente. Al tratarse de una cola secuencial, la tasa máxima de entrada estará limitada por el nodo más restrictivo, es decir, por el nodo que tenga la mayor tasa mínima de llegadas. Si simulamos con un tiempo entre llegada menor que el asumible por cualquiera de los algoritmos, el tiempo de espera tenderá a infinito y por lo tanto no podremos concluir un resultado óptimo. De esta manera, para determinar el mínimo tiempo entre llegadas con el que podemos simular la red, deberemos obtener primero la lambda mínima de todos los algoritmos y decantarnos por la máxima de éstas para que el sistema no tienda a infinito

4.1.1 Asignación del número de núcleos de procesamiento a cada algoritmo

Como ya se ha explicado anteriormente el número de núcleos de procesamiento que se le asignará a cada algoritmo viene dado por la relación entre el tiempo de servicio de cada uno de ellos. En la siguiente tabla vemos como se distribuyen los servidores en la placa ODROID:

Tabla 4.1. Distribución del número de núcleos de procesamiento

	Quality	**HotSpot**	**Georef**	**Fusion**
Tiempo de ejecución	0.55 s	0.03 s	0.003 s	1.25 s
Número de cores	2 cores	1 core	1 core	4 cores

La distribución de los núcleos es proporcional al tiempo de ejecución de cada algoritmo, es decir, vemos que el tiempo que necesita la Fusión para ejecutarse es prácticamente el doble que el Quality, por lo que debe proporcionar más servidores al que más lo necesite. Al mismo se observar como el Hotspot y Georef tienen un tiempo de ejecución muy inferior a los otros por lo que requerirán menos servidores para procesar la información. De esta manera llegamos a la conclusión que vemos en la tabla.

No obstante, el hecho de asignar un número diferentes de servidores a cada algoritmo, implica que la tasa entre llegadas de los paquetes se reduzca en consideración, ya que al tener más núcleos de procesamiento el sistema trabaja balanceando la carga entre ellos y reduciendo el tiempo. Para ello, se define el concepto de 1/lambda_inf, que es el tiempo medio entre llegadas consecutivas que limita el sistema para que no tienda a infinito. A partir, de la siguiente fórmula (factor de utilización) es posible calcular este tiempo:

$$\rho = \frac{\lambda inf}{k * \mu} \qquad \textbf{(4.1)}$$

Suponiendo que el sistema está lleno se puede obtener el tiempo que nos va a limitar el sistema en función del número de servidores que se asignen a cada algoritmo. Se pueden ver reflejados los resultados de estos tiempos en la siguiente tabla, siendo *k* el número de núcleos:

Tabla 4.2. Valor límite del tiempo de llegada para cada algoritmo

	Tiempo (s)	Cores	1/lambda_inf (s)
Quality	0.555	2	0.277
Hotspot	0.036	1	0.036
Georef	0.003	1	0.003
Fusion	1.254	4	0.314
Total	1.848	8	0.314

Una vez llegados a este punto, notar que a la hora de realizar las simulaciones pertenecientes a cada algoritmo, como se ha visto en el capítulo 2, en este punto sigue interesando saber el comportamiento del tiempo de espera en relación al tiempo entre llegadas. Por lo tanto, se utiliza la interfaz por línea de comandos, que nos permite realizar simulaciones por lotes, viendo así los valores obtenidos en más de una simulación de una misma ejecución.

4.1.3 Análisis independiente utilizando un modelo de colas general

Una vez asignamos el número de servidores a cada algoritmo, debemos encontrar la lambda mínima de cada uno de ellos. Para esto, se ha realizado el análisis independiente para a posteriori limitar la red de colas con la mayor lamba mínima encontrada.

Siguiendo la línea de todo el documento, vamos a focalizar en uno de ellos, en concreto, en el Quality (el resto de algoritmos pueden encontrarse en el Anexo). Después de ajustar su distribución, como se ha visto en el punto 3, vemos las siguientes líneas de código para la realización de la simulación individual:

```
[General]
cmdenv-express-mode = true
output-scalar-file = ${resultdir}/${configname}-${runnumber}.sca
output-vector-file = ${resultdir}/${configname}-${runnumber}.vec
sim-time-limit = 50000s
warmup-period = 25s

[Config queue_net]
network = queue_net
repeat = 1
**.serviceTime = gamma_d(1887.224,0.000293s )
**.source.interArrivalTime = (${0.275..0.295 step 0.001 }s)
**.maxQueueLength = 100000
**.numServers = 2
```

Centrándonos de la configuración de la cola podemos ver como el tiempo de servicio le atribuimos la condición de ejecutarse como una distribución gamma, pasándole los valores de los parámetros shape y scale (en segundos), que se obtuvieron en el apartado 3 de este documento. Del mismo modo, no le atribuimos una distribución exponencial, sino de manera constante, al tiempo

entre llegadas de los paquetes, estudiamos ante todo los puntos más críticos donde el tiempo medio de espera en cola pasa de infinito a cero, es decir la zona de trabajo efectiva, con los saltos que creamos oportunos para el buen análisis y de esta manera representar gráficamente los resultados obtenidos. Además, añadimos una cola que tienda a infinito y como hemos visto anteriormente dos servidores que se definen dentro del módulo *queueAndServers_modified* para procesar los flujos generados por la fuente. El archivo NED quedaría de la siguiente manera:

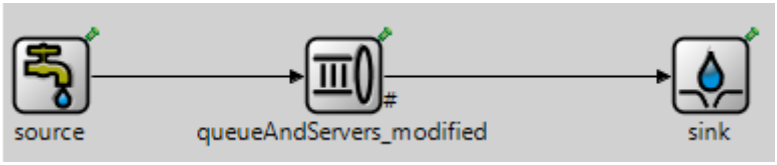

Figura 4.1 Estructura del modelo de colas mediante el fichero NED

Una vez concluye la simulación con OMNeT++, se trasladan los datos obtenidos a R, principalmente los valores de tiempo medio de espera en cola, para poder analizar de manera gráfica los resultados obtenidos. El resultado que obtenemos es el siguiente:

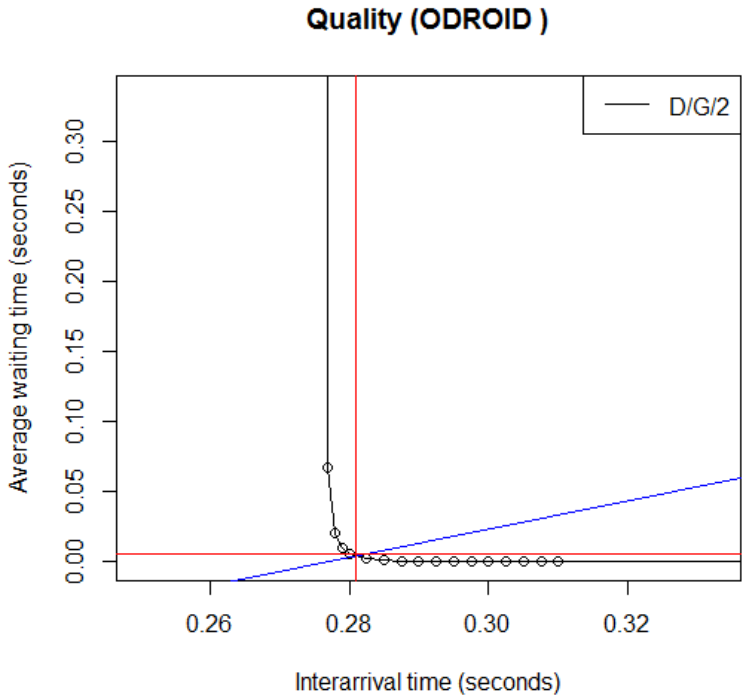

Figura 4.2 Valor mínimo para el tiempo entre llegadas del algoritmo Quality con un modelo de colas D/G/2

La imagen muestra el tiempo medio de espera en cola en función del tiempo entre llegadas utilizando el modelo de colas D/G/2 para el algoritmo Quality. La

asíntota vertical en 1/lambda_inf se corresponde con la misma que utilizando un sistema de colas M/M/2 (*Figura 2.4)*. La diferencia está en la forma de la curva, donde al utilizar una distribución general hace que se produzca un cambio mucho más brusco.

Para determinar un límite inferior del tiempo entre llegadas, que nos permita mantener el tiempo de espera en cola pequeño sin entrar en la zona en que la curva tiende rápidamente a infinito, se propone buscar el punto de corte entre la curva y una recta (de color azul en la gráfica) con pendiente 1 y que corte el eje de abcisas en la asíntota 1/lambda_inf.

La recta se ha trazado a partir de la siguiente fórmula:

$$Wq = a * \frac{1}{lambda} + b, \ dónde\ b = -a * \frac{1}{lambda\_inf} \quad \textbf{(4.2)}$$

Donde a es igual a 1. De este modo, conseguimos desplazar la recta para que corte en x= $\frac{1}{lambda\_inf}$. Éste es el punto donde el sistema empieza a tender a infinito, por lo tanto encontrando el punto donde se cruce con la curva de trabajo, podemos decir que esa marca es la frecuencia a la que debe trabajar el sistema para que no entre en saturación y a la vez se mantenga estable.

El punto de corte de las dos rectas rojas marca el punto donde se determina el límite, que llamaremos 1/lambda_min. En este caso, 1/lambda_mín es igual a 0.281 segundos y es la que limitará el sistema.

Para establecer el tiempo 1/lambda_min que vamos a usar en la red global, se ha ejecutado el mismo procedimiento para todos los algoritmos. Siendo la siguiente tabla dónde se muestran los tiempos de todos ellos y pudiendo así mostrar cuál es el tiempo máximo que marcará la tasa límite para que el sistema trabaje de una forma óptima y que al mismo tiempo no entre en saturación. También se muestra el tiempo de Jellyfish, aunque se utilizará como otro caso de uso.

Tabla 4.3. Valores mínimos del tiempo de llegada de cada algoritmo

	Quality	**Hotspot**	**Georef**	**Fusion**	**Jellyfish**
1/lambda_inf (s)	0.277	0.036	0.00347	0.314	0.974
1/lambda_min (s)	0.281	0.037	0.00355	0.316	0.989

De este modo podemos confirmar que el tiempo que limita el sistema para la misión de los Hotspots es el que viene marcado por la Fusion.

4.1.4 Análisis independiente utilizando un modelo de colas exponencial

En el siguiente apartado, uno de los puntos que se va a mostrar son las diferencias que existen entre utilizar un modelo de colas exponencial o utilizar un modelo de colas con distribuciones generales. Para ello, es necesario que del mismo modo que en el apartado 4.1.3, se analicen los tiempos de cada algoritmo y que uno de ellos marcará la limitación del sistema.

La forma de extraer estos valores es la misma que en el punto anterior, no obstante ahora se utiliza distribuciones exponenciales, es decir, hay que indicar en OMNeT que tanto el tiempo de servicio como el tiempo medio entre llegadas de las imágenes se produzca de forma exponencial. De la misma manera el algoritmo empleado ha sido Quality y el código en OMNeT++ es el siguiente:

```
[General]
cmdenv-express-mode = true
output-scalar-file = ${resultdir}/${configname}-${runnumber}.sca
output-vector-file = ${resultdir}/${configname}-${runnumber}.vec
sim-time-limit = 50000s
warmup-period = 25s
#[Config queue_net]
network = queue_net
repeat = 1
**.serviceTime = exponential(0.5548s)
**.source.interArrivalTime = exponential(${0.5..1 step 0.1}s)
exponential(${0.1..2.8 step 0.15}s)
**.maxQueueLength = 100000
**.numServers = 2
```

Como se puede ver se realiza la simulación por línea de comandos, ya que se marca un rango de simulaciones y nos reflejará los valores obtenidos en más de una simulación en una misma ejecución. A diferencia de las simulaciones que se han visto en el capítulo 2, a estas simulaciones se les ha distribuido el número correcto de servidores para poder realizar a posterior una comparativa en igualdad de condiciones.

Una vez obtenidos los valores, se han cargado a través de un archivo CSV en R, para ser capaces de mostrarlos gráficamente. El resultado que ha generado ha sido el siguiente:

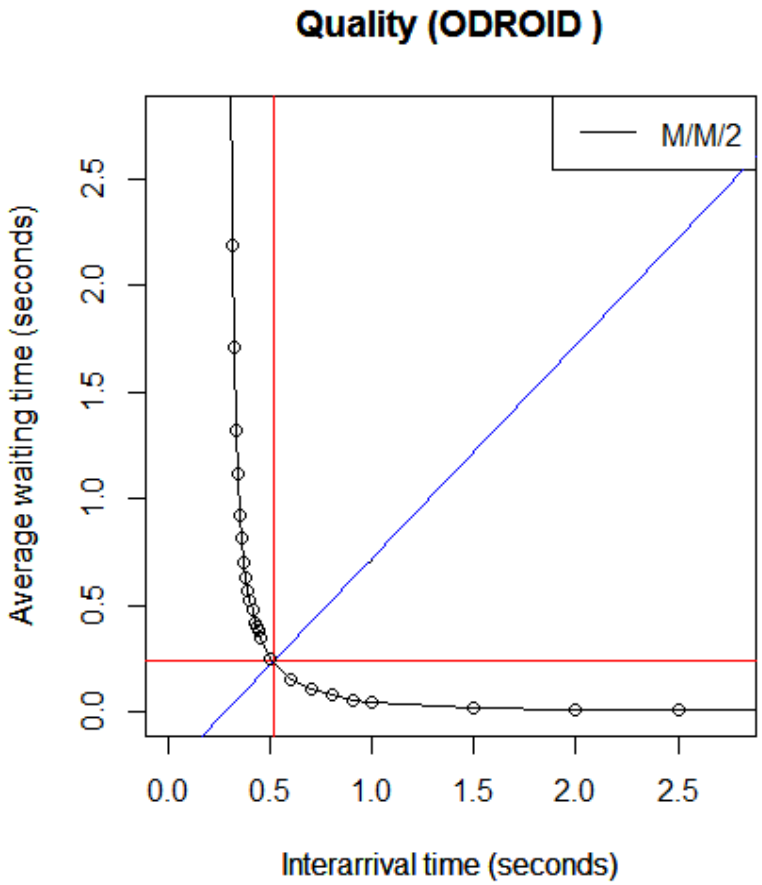

Figura 4.3 Valor mínimo para el tiempo entre llegadas para el algoritmo Quality utilizando un modelo de cola M/M/2

Se puede observar que el hecho de utilizar dos servidores, hace que el tiempo entre llegada de las imágenes sea inferior que si se utiliza un servidor. Esto se puede considerar positivamente ya que permitirá el ingreso de más imágenes por segundo al sistema. Como en el caso anterior se propone una heurística para determinar el límite inferior del tiempo entre llegadas. Es en el punto de corte entre las dos líneas rojas coincidiendo con la heurística donde se determina el valor mínimo del tiempo entre llegadas.

Para hacer más llevadera la memoria, el resto de gráficos se pueden ver en el anexo. No obstante, a continuación se muestra una información semejante a la *tabla 4.3* del punto anterior, pero en este caso, hace referencia al modelo de cola siguiendo una distribución exponencial, mostrando el tiempo mínimo entre llegadas de cada algoritmo y decidiendo del mismo modo el tiempo que limita el sistema.

Tabla 4.4 Valores mínimos de cada algoritmo utilizando un modelo de colas exponencial

	Quality	**Hotspot**	**Georef**	**Fusion**	**Jellyfish**
Lambda_inf (s)	0.277	0.0356	0.00347	0.314	0.974
Lambda_min (s)	0.515	0.0715	0.00714	0.535	1.5

Viendo los resultados mostrados en la tabla, podemos ver que utilizando un modelo de colas exponencial es la Fusion la que indica la frecuencia máxima que se debe emplear para un funcionamiento correcto del sistema.

4.1.5 Análisis global para el caso de uso HotSpot

Después de ver cómo se comporta el sistema en el caso del Quality, vamos ver la conducta de todo el sistema (el resto de gráficos se pueden ver en el Anexo).

En la siguiente imagen vemos el archivo NED generado en OMNeT++, en el cual describimos la estructura del modelo de colas que se ha llevado a cabo:

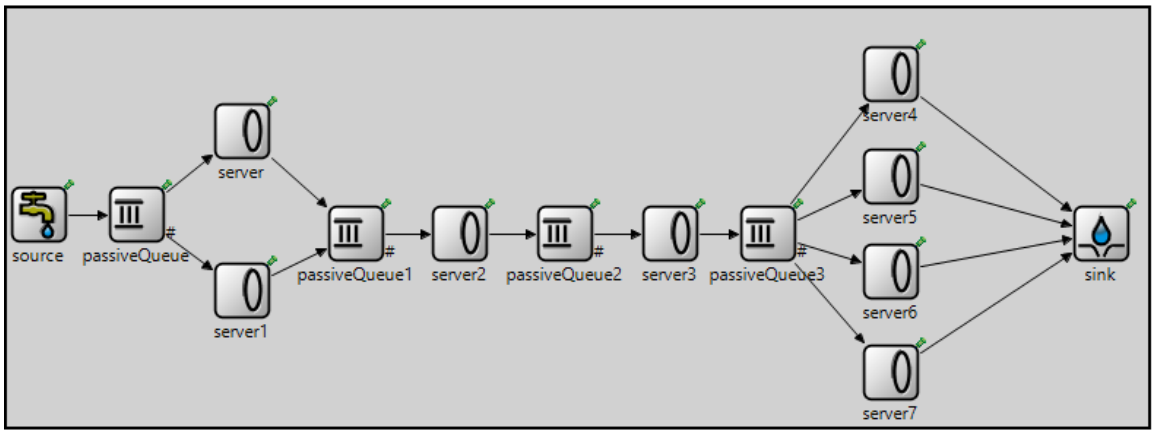

Figura 4.4 Estructura del modelo de colas empleado para el caso de uso Hotspot**.**

El código empleado para la colocación y conexión de todos los elementos es la siguiente:

```
import org.omnetpp.queueing.PassiveQueue;
import org.omnetpp.queueing.QueueAndServers_modified;
import org.omnetpp.queueing.QueueNoServers;
import org.omnetpp.queueing.ResourceBasedQueue_mod;
import org.omnetpp.queueing.Router;
import org.omnetpp.queueing.Server;
import org.omnetpp.queueing.Sink;
import org.omnetpp.queueing.Source;


network All_queue_net
{
    @display("bgb=841,290");
    submodules:
        source: Source {
            @display("p=22,141");
        }
        sink: Sink {
            @display("p=749,154");
        }
```

```
        server: Server {
            @display("p=161,84");
        }
        server1: Server {
            @display("p=161,193");
        }
        server2: Server {
            @display("p=310,154");
        }
        passiveQueue: PassiveQueue {
            @display("p=89,141");
        }
        passiveQueue1: PassiveQueue {
            @display("p=239,154");
        }
        passiveQueue2: PassiveQueue {
            @display("p=382,154");
        }
        server3: Server {
            @display("p=454,154");
        }
        passiveQueue3: PassiveQueue {
            @display("p=521,154");
        }
        server4: Server {
            @display("p=605,45");
        }
        server5: Server {
            @display("p=605,115");
        }
        server6: Server {
            @display("p=605,180");
        }
        server7: Server {
            @display("p=605,249");
        }
    connections:
        source.out --> passiveQueue.in++;
        passiveQueue.out++ --> server.in++;
        passiveQueue.out++ --> server1.in++;
        server.out --> passiveQueue1.in++;
        server1.out --> passiveQueue1.in++;
        passiveQueue1.out++ --> server2.in++;
        server2.out --> passiveQueue2.in++;
        passiveQueue2.out++ --> server3.in++;
        server3.out --> passiveQueue3.in++;
        passiveQueue3.out++ --> server4.in++;
        passiveQueue3.out++ --> server5.in++;
        passiveQueue3.out++ --> server6.in++;
        passiveQueue3.out++ --> server7.in++;
        server4.out --> sink.in++;
        server5.out --> sink.in++;
        server6.out --> sink.in++;
        server7.out --> sink.in++;
}
```

Una vez tenemos montado el sistema mediante el fichero NED, lo siguiente es configurar el fichero .ini de OMNeT y configurar los parámetros que se van a

utilizar. Ahora sí, será necesario establecer un valor fijo del tiempo entre llegadas. Este valor debe ser el máximo 1/lambda_min encontrado entre todos los algoritmos. Como vemos reflejado en la *tabla 4.3* debemos escoger la tasa límite marcada por la fusión, concretamente 0.3155 segundos. Es la fuente la encargada de generar este flujo que al venir dado a la frecuencia marcada, vamos a ver que en cualquier punto del sistema los servidores trabajan sin que estos entren en colapso en ningún momento. En este caso se ha procedido a la simulación, mediante la interfaz de usuario gráfico, la cual nos deja hacer una simulación en una misma ejecución, en este caso tomando únicamente un valor de lambda. A continuación se muestra la parte del código dónde se establecen los parámetros principales:

```
[General]
cmdenv-express-mode = true
#cmdenv-interactive=true
output-scalar-file = ${resultdir}/${configname}-${runnumber}.sca
output-vector-file = ${resultdir}/${configname}-${runnumber}.vec
sim-time-limit = 50000s #fusion
warmup-period = 25s #fusion

[Config All_queue_net]
network = All_queue_net
repeat = 1

**.server.serviceTime = gamma_d(1887.224,0.000293s)
**.server1.serviceTime = gamma_d(1887.224,0.000293s)
**.server2.serviceTime = gamma_d (185.747, 0.000192s)
**.server3.serviceTime = lognormal(-5.66306641,0.02906538)*1s
**.server4.serviceTime = lognormal(0.226627, 0.00695668)*1s
**.server5.serviceTime = lognormal(0.226627, 0.00695668)*1s
**.server6.serviceTime = lognormal(0.226627, 0.00695668)*1s
**.server7.serviceTime = lognormal(0.226627, 0.00695668)*1s

**.source.interArrivalTime = (0.3155s)
```

Es importante recordar, que en el capítulo 3 ajustábamos los parámetros de la distribución de probabilidad del tiempo de servicio para definir como se modelará el flujo de los servidores en cada algoritmo, teniendo en cuenta que a cada uno se le asigna un número diferente de servidores. De este modo, con las líneas de código que aparecen en la imagen, se le imputa a cada servidor la distribución elegida y con los parámetros ajustados a cada uno de ellos. Además, se define el tiempo entre llegadas con el valor de la fusión escogido anteriormente.

Al finalizar la simulación, se puede entrar a valorar los resultados en diferentes puntos del sistema. De la misma manera que se lleva haciendo a lo largo de este documento, a continuación se muestran los resultados obtenidos en la cola del Quality:

Module	Name	Value
All_queue_net.pas...	queueingTime:mean	NaN
All_queue_net.pas...	queueingTime:max	NaN
All_queue_net.pas...	queueLength:timeavg	NaN
All_queue_net.pas...	queueLength:max	NaN
All_queue_net.pas...	dropped:count	0.0

Figura 4.5 Resultados obtenidos en la cola del algoritmo Quality

Los resultados que se muestran pueden parecer sorprendentes a primera vista. El hecho que no aparezca ningún valor quiere decir que su valor se aproxima a 0. Esto viene dado a que el tiempo entre llegadas que limita la cola del Quality es menor a la del sistema, y por lo tanto cuando su dos servidores estén en ejecutándose atenderán a los clientes sin formar colas en el sistema, ya que su 1/lambda_min es de 0.281 segundos.

A continuación, se muestra la figura con los valores obtenidos en la cola de la Fusion:

Module	Name	Value
All_queue_net.pas...	queueingTime:mean	0.015862603138363
All_queue_net.pas...	queueingTime:max	0.090114103453
All_queue_net.pas...	queueLength:timeavg	0.025498789151467
All_queue_net.pas...	queueLength:max	1.0
All_queue_net.pas...	dropped:count	0.0

Figura 4.6 Resultados obtenidos en la cola del algoritmo Fusion

En este punto, como la simulación viene marcada por la frecuencia mínima que marca la Fusion muestra el estado en el que se encuentra su cola. Los datos confirman que la simulación es idónea. Por un lado muestra que la longitud de la cola va a ser como máximo de 1. Recordemos que el tiempo de ejecución de la Fusion es 1,25 segundos, al tener 4 servidores que distribuyen la carga que les llega a una frecuencia de 0.331 segundos, cuando los cuatros servidores estén atendiendo al cliente, el siguiente paquete que llegue lo servirá de nuevo el primer servidor, que ya habrá acabado de atender a su cliente y el paquete que esté en la cola estará un tiempo mínimo esperando, tanto es así que cuando llegue el siguiente paquete éste ya estará atendido por el primer servidor. Esto es exactamente lo que vemos en la imagen, ya que nos muestra que el tiempo medio de espera en cola en este punto del sistema va a ser de 0.01586 segundos con un tiempo máximo de espera del 0.09 segundos. Esto va a permitir que no se pierda ningún paquete ya que los servidores al acabar de atender a su respectivo cliente, pasarán al siguiente sin permitir formar colas.

Ahora bien, si se quiere sacar una conclusión final del comportamiento del sistema, hay que proceder a un análisis en el punto final de la red. Para ello, se

deben escoger los datos resultantes en el módulo llamado *sink.* Es en la siguiente imagen, dónde se muestran los datos, proporcionados por el fichero ANF:

Module	Name	Value
All_queue_net.sink	totalServiceTime:mean	1.846531717106
All_queue_net.sink	totalQueueingTime:mean	0.0080448097706291
All_queue_net.sink	arrivalRate:mean	3.1694354644773

Figura 4.7 Resultados obtenidos del sistema

Una vez llegados a este punto, los datos que se obtienen son medias de los parámetros importantes que permitirán sacar conclusiones. Ahora lo que nos muestra el sistema en primer lugar es el tiempo medio de servicio. Esto lo habíamos calculado anteriormente, ya que este parámetro al fin y al cabo es el tiempo total que tarda el cliente en ser atendido por todos los servidores por los que ha pasado, ya que se trata de un sistema multifase donde existe un conjunto de sistemas interconectados. Por otro lado, muestra la tasa de llegada del sistema. Este parámetro es el número de entidades promedio que ingresan al sistema por unidad de tiempo, es decir, cada 0.315 segundos llega al sistema una imagen, esto quiere decir que cada segundo se procesan 3.169 imágenes. Un valor a tener en cuenta y que se comparará con el caso de un sistema siguiendo una distribución exponencial para ver la importancia de ello. De esta manera, al ser la tasa de servicio del sistema menor que la tasa de llegada esto hace que no se colapse nunca y que el tiempo medio de espera en cola del sistema sea de 0.008 segundos, es decir prácticamente se está sirviendo a los clientes al mismo tiempo que llegan al servidor.

Por lo tanto, en cualquier punto del sistema los servidores serán capaces de procesar los datos sin formar colas que provoquen la pérdida de paquetes. Esto es un punto a remarcar, ya que otra opción sería ingresar al sistema más imágenes pero esto provocaría que la red se colapsara produciendo la pérdida de paquetes.

Por último, también se ha realizado la simulación para este caso de uso pero siguiendo una distribución exponencial. Una vez obtenidos los valores de esta simulación, podremos sacar una conclusión. Del mismo modo, como se ha visto en el punto 4.1.4 se ha puesto en práctica las simulaciones pertinentes para cada algoritmo. En este, caso, el tiempo que limita el sistema sigue siendo el de la Fusion. El hecho de que su tiempo de servicio sea el más grande, aunque se le asignen 4 servidores, hace que sea el encargado de marcar el límite para que el sistema no entre en saturación.

Como muestra la *tabla 4.4* el tiempo entre llegadas que limita el sistema es la fusión, pero eso sí, en este caso la tasa de ingreso de las imágenes al sistema es menor que si se usa una distribución real. El hecho que el valor de 1/lambda_min sea mayor en el caso exponencial genera que las imágenes tarden más en ingresar al sistema. De este modo, podemos decir que

trabajando con estas distribuciones no somos del todo realistas a la hora de modelar el sistema. El código en el fichero de configuración es el siguiente:

```
[General]
cmdenv-express-mode = true
output-scalar-file = ${resultdir}/${configname}-${runnumber}.sca
output-vector-file = ${resultdir}/${configname}-${runnumber}.vec
sim-time-limit = 50000s #fusion
warmup-period = 25s #fusion

[Config All_queue_net]
network = All_queue_net
repeat = 1

**.server.serviceTime = exponential(0.5548s)
**.server1.serviceTime = exponential(0.5548s)
**.server2.serviceTime = exponential(0.035666s)
**.server3.serviceTime = exponential(0.003473s)
**.server4.serviceTime = exponential(1.254392s)
**.server5.serviceTime = exponential(1.254392s)
**.server6.serviceTime =exponential(1.254392s)
**.server7.serviceTime = exponential(1.254392s)

**.source.interArrivalTime = exponential(0.535s)
```

Se puede observar cómo se le asignan distribuciones exponenciales a todos los servidores con sus respectivos tiempos de servicio y del mismo modo al tiempo entre llegadas de las imágenes.

Nos encontramos con el mismo caso que antes en el que depende el punto dónde analicemos el sistema sacaremos unas conclusiones u otras. Si paramos a mirar la cola del Quality vemos lo siguiente:

Module	Name	Value
All_queue_net.pas...	queueingTime:mean	0.56783681934142
All_queue_net.pas...	queueingTime:max	4.121793536893
All_queue_net.pas...	queueLength:timeavg	0.37771117705675
All_queue_net.pas...	queueLength:max	14.0
All_queue_net.pas...	dropped:count	0.0

Figura 4.8 Resultados obtenidos en la cola del algoritmo Quality siguiendo un modelo de colas exponencial

Observamos en este caso, que el tiempo medio de espera en cola que se ha obtenido es de 0.567 segundos con picos máximos de 4.121 segundos, esto va a provocar que desde que se inserten las imágenes en el sistema se formen colas. Esto viene producido porque el tiempo mínimo que se ha calculado es muy similar al de la Fusion que es el que limita el sistema. Podemos ver que las imágenes se pasarán un tiempo medio de 0.377 segundos en cola llegando en un caso máximo a tener 14 imágenes en cola. En el caso que se ha

simulado hemos valorado una cola infinita por eso no se producirán pérdidas de imágenes.

Como el caso de la Fusion es muy similar a este, pasamos a ver los resultados en la *sink*, es aquí donde se recogen los valores medios del sistema para hacernos una idea de cómo ha ido la transmisión:

Module	Name	Value
All_queue_net.sink	totalServiceTime:mean	1.8475068124899
All_queue_net.sink	totalQueueingTime:mean	0.39881048581012
All_queue_net.sink	arrivalRate:mean	1.8684189091922

Figura 4.9 Resultados obtenidos del sistema usando un modelo de cola exponencial.

Vemos que el tiempo de servicio del sistema es el mismo empleando distribuciones exponenciales que generales, esto es un parámetro que nos indica que las simulaciones son correctas. No obstante, como era de esperar, obtenemos un tiempo medio de espera en cola bastante más grande que en el caso anterior, a pesar de estar ingresando 1.86 imágenes por segundo, mientras que en el caso general el número de imágenes por segundo que ingresan al sistema es de 3.169.

Una de las conclusiones que se sacan después de ver ambas simulaciones, es que al trabajar con un modelo de colas exponencial, a la hora de extraer el valor mínimo de la tasa entre llegadas de las imágenes necesario para que no se formen colas, estaríamos obteniendo un valor muy por encima del que realmente nos permite el sistema.

4.2 Escenario 2 – Caso de uso Jellyfish

Como hemos visto en puntos anteriores, Jellyfish también es un algoritmo de segmentación, pero a diferencia del caso de uso Hotspot, éste es un único algoritmo con diferentes fases. No obstante, en este capítulo nos centraremos de la misma manera que en el caso anterior, en encontrar la tasa mínima permitida que nos permita trabajar en un área efectiva. Para ello, después del ajuste de parámetros de la distribución de probabilidad del tiempo de servicio, y seleccionar aquella distribución que más se ajuste, se han realizado las simulaciones pertinentes para obtener la tasa máxima de llegadas que nos permite mantener un tiempo de espera mínimo, manteniendo la estabilidad del sistema.

En este caso, al tener únicamente un algoritmo, encontrando la frecuencia mínima con la que llegan los paquetes entre sí, marca el límite del sistema para

que no entre en saturación y de permitir de este modo que el sistema funcione correctamente transmitiendo el número máximo de imágenes.

A continuación se muestra el fichero NED y el código empleado para describir el sistema de Jellyfish:

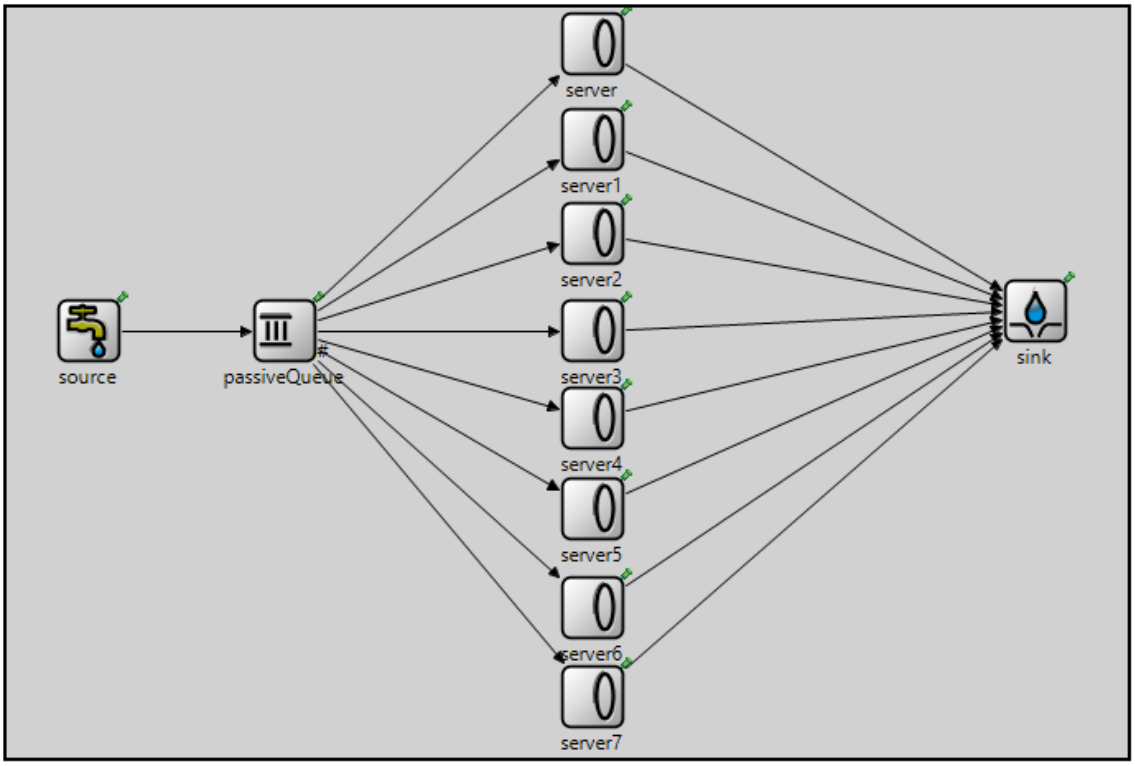

Figura 4.10 Estructura del modelo de colas empleado para Jellyfish

Las líneas de código que generan los enlaces que unen cada módulo del modelo mostrado son las siguientes:

```
import org.omnetpp.queueing.PassiveQueue;
import org.omnetpp.queueing.QueueAndServers_modified;
import org.omnetpp.queueing.Server;
import org.omnetpp.queueing.Sink;
import org.omnetpp.queueing.Source;

network queue_net
{
    @display("bgb=681,455");
    submodules:
        source: Source {
            @display("p=50,196");
        }
        sink: Sink {
            @display("p=625,184");
        }
```

```
        server: Server {
            @display("p=356,22");
        }
        passiveQueue: PassiveQueue {
            @display("p=169,196");
        }
        server1: Server {
            @display("p=356,80");
        }
        server2: Server {
            @display("p=356,137");
        }
        server3: Server {
            @display("p=356,196");
        }
        server4: Server {
            @display("p=356,249");
        }
        server5: Server {
            @display("p=356,304");
        }
        server6: Server {
            @display("p=356,364");
        }
        server7: Server {
            @display("p=356,418");
        }
    connections:
        source.out --> passiveQueue.in++;
        server.out --> sink.in++;
        passiveQueue.out++ --> server.in++;
        passiveQueue.out++ --> server1.in++;
        passiveQueue.out++ --> server2.in++;
        passiveQueue.out++ --> server3.in++;
        passiveQueue.out++ --> server4.in++;
        passiveQueue.out++ --> server5.in++;
        passiveQueue.out++ --> server6.in++;
        server1.out --> sink.in++;
        server2.out --> sink.in++;
        server3.out --> sink.in++;
        server4.out --> sink.in++;
        server5.out --> sink.in++;
        server6.out --> sink.in++;
        passiveQueue.out++ --> server7.in++;
        server7.out --> sink.in++;
}
```

Se establece en el sistema un módulo *source* que es el encargado de generar el flujo, seguido del módulo *pasive_queue* y ocho servidores dónde se definen los parámetros importantes como son el tiempo de servicio (con una distribución lognormal), tasa de llegadas entre paquetes (1/lambda mínima), que ha sido encontrada a partir de R cómo se explicará a continuación o el número de servidores que se ejecutarán en el algoritmo. Referente a este último punto, cabe decir que es más simple asignar un número de cores que en el caso de uso de Hotspot, ya que al ser un único algoritmo se le asignarán todos ellos. Sabiendo que la placa ODROID tiene disponibles 8 cores y que por

lo tanto todos irán destinados a Jellyfish para el procesamiento a tiempo real de las imágenes.

Antes de mostrar los resultados obtenidos, donde se verá la tasa mínima de llegadas entre paquetes, es necesario observar la configuración que se ha utilizado tanto en OMNeT++ como en R para llegar a obtener lo valores de los parámetros para la transmisión idónea.

La configuración del fichero .ini es la siguiente:

```
[General]
cmdenv-express-mode = true
output-scalar-file = ${resultdir}/${configname}-${runnumber}.sca
output-vector-file = ${resultdir}/${configname}-${runnumber}.vec
sim-time-limit = 1000000s
warmup-period = 500s

[Config queue_net]
network = queue_net
repeat = 1

**.server.serviceTime = lognormal (2.05324531, 0.01474755)*1s
**.server1.serviceTime = lognormal (2.05324531, 0.01474755)*1s
**.server2.serviceTime = lognormal (2.05324531, 0.01474755)*1s
**.server3.serviceTime = lognormal (2.05324531, 0.01474755)*1s
**.server4.serviceTime = lognormal (2.05324531, 0.01474755)*1s
**.server5.serviceTime = lognormal (2.05324531, 0.01474755)*1s
**.server6.serviceTime = lognormal (2.05324531, 0.01474755)*1s
**.server7.serviceTime = lognormal (2.05324531, 0.01474755)*1s


**.server.source.interArrivalTime = (${0.97..2 step 0.001}s)
**.maxQueueLength = -1
```

Del mismo modo que en casos anteriores, en primer lugar se ha realizado una simulación por línea de comandos, ya que es útil para realizar simulaciones por lotes. Esto permite reflejar los valores obtenidos en más de una simulación dentro de una misma ejecución. Estos lotes, vienen reflejados por el tiempo entre llegadas marcado por la fuente. Como se pude observar indicamos que la ejecución se produzca de 0.97 segundos a 2 segundos, en saltos de 0.001 segundos de manera arbitraria. En este rango de tiempo se encuentran los valores donde se produce la curva y el sistema trabaja de una forma efectiva, para más adelante encontrar la frecuencia mínima permitida a la que tienen que llegar las imágenes sin que el sistema se sature. También se han realizado simulaciones en valores superiores e inferiores, viendo que contra más pequeño sea este valor el tiempo medio de espera en cola tiende a infinito y en caso contrario los valores se aproximan a cero.

Por otro lado, le indicamos que el tiempo de servicio de cada núcleo de procesamiento utilice una distribución lognormal con sus respectivos parámetros y además vemos como se le asigna los ocho servidores disponibles.

A partir de los resultados obtenidos se ha generado un archivo CSV, que se carga con R y nos proporcionará el gráfico en cuestión. Este gráfico podemos verlo en los anexos. De la misma manera que en el caso de uso de HotSpot, vamos a ver cómo se comporta el sistema utilizando el tiempo que limita el sistema y que hemos encontrado en la *tabla 4.3*. Ahora utilizamos la interfaz de usuario gráfico para realizar una simulación tomando únicamente el valor fijado de 1/lambda_min, con la siguiente línea:

```
**.source.interArrivalTime = (0.989s)
```

Después de realizar la simulación, en primer lugar analizamos los valores obtenidos en la cola. Los resultados son los siguientes:

Module	Name	Value
queue_net.passiveQueue	queueingTime:mean	0.073336401451605
queue_net.passiveQueue	queueingTime:max	0.695518720164
queue_net.passiveQueue	queueLength:timeavg	0.014190206657665
queue_net.passiveQueue	queueLength:max	1.0
queue_net.passiveQueue	dropped:count	0.0

Figura 4.11 Resultados obtenidos en la cola de Jellyfish

Los valores que se recogen en la cola nos muestran que el tiempo medio de espera en cola es de 0.073 segundos con picos que ascienden hasta 0.695 segundos. El número medio de imágenes en cola es 0.141, y lo que es más importante, la longitud máxima de la cola no sobrepasa la unidad, y por lo tanto la implementación de una pequeña cola sería suficiente para asegurar que el sistema es capaz de procesar las imágenes a la frecuencia de adquisición y sin perder ninguna imagen.

Después de ver los resultados obtenidos siguiendo un modelo de colas real, ahora se muestran los valores si lo hiciéramos con una cola que sigue un modelo exponencial.

La estructura de modelo seguida en el fichero NED ha sido la misma para ambos casos. El código en OMNeT que se ha empleado para generar estos resultados ha sido el siguiente:

```
[General]
cmdenv-express-mode = true
output-scalar-file = ${resultdir}/${configname}-${runnumber}.sca
output-vector-file = ${resultdir}/${configname}-${runnumber}.vec
sim-time-limit = 1000000s
warmup-period = 500s

[Config queue_net]
network = queue_net
repeat = 1

**.server.serviceTime = exponential (7.794s)
```

```
**.server1.serviceTime = exponential (7.794s)
**.server2.serviceTime = exponential (7.794s)
**.server3.serviceTime = exponential (7.794s)
**.server4.serviceTime = exponential (7.794s)
**.server5.serviceTime = exponential (7.794s)
**.server6.serviceTime = exponential (7.794s)
**.server7.serviceTime = exponential (7.794s)

**.source.interArrivalTime = exponential (${0.875..4 step 0.125}s)
**.maxQueueLength = -1
```

A diferencia del caso anterior, se refleja como ahora se utilizan distribuciones exponenciales para todos los servidores, empleando el mismo tiempo de ejecución. De la misma manera se le asigna al tiempo entre llegadas una distribución exponencial, realizando en este caso varias simulaciones dentro del rango marcado. También se han realizado simulaciones fuera de estos rangos para ver cómo se comporta el sistema.

Acto seguido, una vez recogidos los valores resultantes se han cargado mediante R para generar el gráfico en cuestión y sacar a partir de aquí el tiempo mínimo entre llegadas que limita el sistema. (Se puede ver en el anexo).

Una vez obtenido este valor, se ha cambiado el siguiente para ver el comportamiento de la red de forma global.

```
**.source.interArrivalTime = exponential (1.5s)
```

Después de realizar la simulación los valores mostrados en la cola son los siguientes:

Module	Name	Value
queue_net.passiveQueue	queueingTime:max	32.460954482563
queue_net.passiveQueue	queueLength:max	26.0
queue_net.passiveQueue	queueingTime:mean	2.8045659165655
queue_net.passiveQueue	queueLength:timeavg	0.36459137679094
queue_net.passiveQueue	dropped:count	0.0

Figura 4.12 Resultados obtenidos en la cola de Jellyfish con un sistema de colas exponencial

En este punto del sistema observamos cómo no se está siendo muy eficaz, ya que el tiempo medio de espera en cola es de 2.8 segundos, un valor mucho mayor que si modelamos usando una distribución general. Además la longitud de esta cola puede ascender hasta 26 con picos máximos de espera de hasta 32.46 segundos, por lo que habría que implementar una cola bastante grande para asegurar que el sistema es capaz de soportar el procesamiento de las imágenes sin producirse pérdidas.

Por lo tanto, una vez analizados las dos modelos de colas siguiendo distribuciones diferentes, queda claro que el hecho de utilizar distribuciones generales va a permitir en este caso de uso realizar un modelado más realista, permitiendo el mayor número de ingresos de imágenes por segundo.

CAPÍTULO 5. CONCLUSIONES

En este capítulo se va a ofrecer una síntesis general del trabajo realizado con una posterior conclusión de los resultados obtenidos.

Como se especifica en el resumen del proyecto, estos últimos años el incremento de aviones no tripulados está incrementando considerablemente sobretodo en tareas de teledetección. Para algunas aplicaciones es necesario que los datos que se recogen en el vuelo tengan una respuesta casi inmediata.

Es por tanto necesario, que se realicen investigaciones sobre cómo realizar un modelado más realista. Para ello, mediante el análisis y la simulación nos ha sido posible estimar a qué frecuencia de adquisición podemos trabajar sin saturar el sistema, siendo uno de los objetivos clave del proyecto.

En un primer lugar se ha tratado de encontrar un entorno de trabajo que ha permitido simular redes de colas especificando las distribuciones empleadas tanto en tiempo de servicio como en los tiempos de llegadas. Para ello se han utilizado dos herramientas, OMNeT++ y R. Ambas herramientas con un potencial bastante alto.

En un primer momento se ha cumplido con el objetivo de evaluar el comportamiento de un conjunto de algoritmos con distribuciones exponenciales con el fin de demostrar que los valores obtenidos son aptos al compararlos con los valores mostrados de forma analítica por R.

Pero es cierto, que si tratamos con distribuciones exponenciales los resultados que se obtienen no son los óptimos para procesado de imágenes a tiempo real. Para ello como se planteaba en uno de los objetivos se han ajustado las funciones de distribución de probabilidad de los tiempos de ejecución de cada algoritmo, consiguiendo de este modo encontrar la distribución más acorde con cada uno de ellos.

Se han evaluado, a través de simulaciones, los comportamientos de los algoritmos de manera individual utilizando cada uno las distribuciones correspondientes. Además, se ha propuesto un mecanismo para determinar la tasa máxima de llegadas que puede soportar un algoritmo distribuyendo el número de núcleos de procesamiento, teniendo en cuenta la placa en la que se desarrollan los algoritmos y el tiempo de servicio que debe emplear cada servidor.

A partir de aquí, se han propuesto dos casos de uso para llevar a cabo determinadas misiones. El primero de ellos es un escenario en el cual se ha planteado una red formada por cuatro de los algoritmos. Cada uno de ellos, con un tiempo entre llegada de los paquetes y un número de núcleos de procesamiento diferentes. Esto ha conllevado a realizar un estudio, en cuál se debía conseguir el tiempo mínimo entre llegadas de los paquetes del algoritmo que más tiempo tardará en procesar la información, ya que este, es el que

limita el sistema. Si se hubiera usado un tiempo inferior al dicho, esto hubiera llevado a cabo que muchas imágenes se perdieran sin conseguir el objetivo nombrado. En este caso de uso se ha visto que el algoritmo que limitaba el sistema era la Fusion.

Además, la estructura del modelo empleado ha permitido formar la red utilizando cada algoritmo sus respectivas distribuciones para sus servidores.

A posterior se ha realizado una comparativa de la red usando un modelo de colas exponencial y otro general. Se ha podido ver, como efectivamente el hecho de trabajar con distribuciones reales nos hace ser más agresivos, consiguiendo insertar en el sistema un número más elevado de imágenes por segundo y teniendo un tiempo de espera medio en cola prácticamente insignificante.

Otro caso de uso que se ha analizado es el del Jellyfish, el cual está pensado para la detección de medusas. En este caso se ha planteado un escenario el cual está formado únicamente por este algoritmo. Del mismo modo que en el caso de uso anterior se ha encontrado el tiempo mínimo entre llegadas de las imágenes, pero en este caso siendo él mismo el limitante del sistema asignándole los 8 núcleos de procesamiento.

En el caso de uso de Jellyfish, también se ha realizado una comparativa con el mismo fin que el anterior. Se ha podido demostrar de este modo, que utilizando una distribución lognormal se consigue un modelado más realista insertando un mayor número de imágenes por segundo que mediante un modelo de colas exponencial.

Como trabajos futuros, por un lado se propone validar los resultados obtenidos, ejecutando las aplicaciones en un entorno real y utilizando la misma placa.

También será interesante automatizar todo el proceso de planificación. Es decir, crear una aplicación que, dada la serie de algoritmos que se quieren ejecutar y conociendo sus tiempos de ejecución en una determinada placa, calcule cual que la distribución de núcleos más óptima y cual sería en ese caso la tasa de adquisición máxima a la que se podría trabajar.

ANEXO 1. AJUSTE DE LA DISTRIBUCIÓN DE PROBABILIDAD DEL TIEMPO DE SERVICIO PARA EL RESTO DE ALGORITMOS

Para que la lectura de la memoria sea más llevadera, se ha creado este punto en el anexo con la intención de mostrar gráficamente los resultados finales mostrados en el capítulo 3. En el transcurso de la memoria se ha podido observar una comparativa de valores simulados frente a valores teóricos con el fin de validar los datos de la simulación para a posterior realizar un ajuste en las distribuciones de probabilidad de los tiempos de servicio que utiliza cada servidor en los diferentes algoritmos. Se han mostrado en el Capítulo 3 los pasos empleados para justificar los resultados finales con sus respectivos gráficos del algoritmo Quality. No obstante, también se ha realizado un análisis más exhaustivo con los demás algoritmos que se ha creído conveniente mostrarlos en este punto para una lectura más cómoda.

1.1 Elección de las distribuciones candidatas

En el capítulo referente a este anexo se explica la estadística descriptiva puede ayudar a elegir las distribuciones candidatas a modelar el tiempo de servicio.

A continuación, se muestran los parámetros obtenidos para todos los algoritmos después de poner en práctica la función *descdist* que es la encargada de proporcionar estadísticas descriptivas como el mínimo, máximo, mediana, etc.

1.1.1 Estadísticas descriptivas para el resto de algoritmos

Del mismo modo que se ha explicado en la memoria, se ha empleado la función *descdist* para el resto de algoritmos. A continuación se muestran los parámetros de cada uno de ello.:

- Para el algoritmo HotSpots el resultado es el siguiente:

```
> descdist(sdata, boot = 1000)
summary statistics
------
min:  0.05368455   max:  0.9463154
median:  0.839104
mean:  0.8490433
estimated sd:  0.1222265
estimated skewness:  -5.784083
estimated kurtosis:  41.22181
```

Figura A1.1 Parámetros de la función *descdist* (HotSpots)

- Los parámetros obtenidos para Georef son los siguientes:

```
> descdist(sdata, boot = 1000)
summary statistics
------
min:  0.3909558   max:  0.6090442
median:  0.4321774
mean:  0.4431526
estimated sd:  0.04059416
estimated skewness:  1.681162
estimated kurtosis:  5.975265
```

Figura A1.2 Parámetros de la función *descdist* (Georef)

- Para el algoritmo Fusion, los valores obtenidos son los mostrados a continuación:

```
> descdist(sdata, boot = 1000)
summary statistics
------
min:  0.01591708   max:  0.9840829
median:  0.2838696
mean:  0.2890356
estimated sd:  0.1404875
estimated skewness:  1.892239
estimated kurtosis:  10.71516
```

Figura A1.3 Parámetros de la función *descdist* (Fusion)

- Por último el caso de Jellyfish nos muestra los siguientes valores:

```
> descdist(sdata, boot = 1000)
summary statistics
------
min:  0.001756741   max:  0.9982433
median:  0.2555155
mean:  0.3158781
estimated sd:  0.2051142
estimated skewness:  1.500079
estimated kurtosis:  5.279779
```

Figura A1.4 Parámetros de la función *descdist* (Jellyfish)

Los valores mostrados nos pueden acercar a elegir qué tipo de distribución será la que mejor se ajuste a cada algoritmo. No obstante, los valores de skewness y kurtosis no son robustos del todo, ya que se calculan con muestras de arranque. Además un skewness diferente de cero nos marca que existe una falta de asimetría de la distribución empírica y el valor de la kurtosis cuantifica el peso de las colas en comparación con la distribución normal. Aun así, estos valores no nos deja claro que distribución es la que más se le ajusta.

Como se explica en la memoria se ha creído conveniente centrar el estudio en las distribuciones exponencial, lognormal, weibull y gamma. Será en el

siguiente punto donde se muestra los resultados obtenidos para la elección de la distribución.

1.2 Parámetros de ajuste de las distribuciones para el resto de algoritmos

Este punto no trata de volver a mostrar lo que ya se ha explicado en la memoria. Pero se ha creído necesario añadir los datos obtenidos de cada algoritmo a partir de las opciones que nos da la función *descdist*. Es por ello que a continuación se va mostrar el *summary* generado para cada algoritmo, en el que nos ofrece de una manera más concisa el ajuste de los parámetros que tiene cada algoritmo para cada distribución.

- El ***summary*** obtenido para HotSpots es el siguiente:

```
> fwe <- fitdist(data, "weibull")
> summary(fwe)
Fitting of the distribution ' weibull ' by maximum likelihood
Parameters :
         estimate   Std. Error
shape 35.63148024 4.9851451325
scale  0.03625448 0.0001511685
Loglikelihood:  261.4896   AIC:  -518.9793   BIC:  -515.1552
Correlation matrix:
          shape     scale
shape 1.0000000 0.6115873
scale 0.6115873 1.0000000

> fln <- fitdist(data, "lnorm")
> summary(fln)
Fitting of the distribution ' lnorm ' by maximum likelihood
Parameters :
           estimate  Std. Error
meanlog -3.33626434 0.011208450
sdlog    0.07925571 0.007919894
Loglikelihood:  222.6201   AIC:  -441.2402   BIC:  -437.4161
Correlation matrix:
              meanlog         sdlog
meanlog  1.000000e+00 -6.307469e-13
sdlog   -6.307469e-13  1.000000e+00

> fex <- fitdist(data, "exp")
> summary(fex)
Fitting of the distribution ' exp ' by maximum likelihood
Parameters :
     estimate Std. Error
rate 28.03827    3.96521
Loglikelihood:  116.6785   AIC:  -231.357   BIC:  -229.445

> fga <- fitdist(data, "gamma")
> summary(fga)
Fitting of the distribution ' gamma ' by maximum likelihood
Parameters :
      estimate Std. Error
shape  185.747   36.97365
```

```
rate  5208.017 1038.06156
Loglikelihood:  226.434   AIC:  -448.8681   BIC:  -445.044
Correlation matrix:
         shape     rate
shape 1.000000 0.998644
rate  0.998644 1.000000
```

- El ***summary*** que muestra los ajustes obtenidos para Georef es el siguiente:

```
> fwe <- fitdist(data, "weibull")
> summary(fwe)
Fitting of the distribution ' weibull ' by maximum likelihood
Parameters :
          estimate Std. Error
shape 26.087799892         NA
scale  0.003529079         NA
Loglikelihood:  749.7795   AIC:  -1495.559   BIC:  -1490.349
Correlation matrix:
[1] NA

> fln <- fitdist(data, "lnorm")
> summary(fln)
Fitting of the distribution ' lnorm ' by maximum likelihood
Parameters :
           estimate  Std. Error
meanlog -5.66306641 0.002906538
sdlog    0.02906538 0.002044295
Loglikelihood:  778.2336   AIC:  -1552.467   BIC:  -1547.257
Correlation matrix:
             meanlog        sdlog
meanlog 1.000000e+00 8.443836e-14
sdlog   8.443836e-14 1.000000e+00

> fex <- fitdist(data, "exp")
> summary(fex)
Fitting of the distribution ' exp ' by maximum likelihood
Parameters :
     estimate Std. Error
rate  287.907   28.79065
Loglikelihood:  466.2638   AIC:  -930.5275   BIC:  -927.9223

> fga <- fitdist(data, "gamma")
> summary(fga)
Fitting of the distribution ' gamma ' by maximum likelihood
Parameters :
        estimate Std. Error
shape   1166.294   16.52221
rate  335782.333 4656.29089
Loglikelihood:  777.4634   AIC:  -1550.927   BIC:  -1545.716
Correlation matrix:
          shape      rate
shape 1.0000000 0.9784142
rate  0.9784142 1.0000000
```

- El ***summary*** para Fusion se muestra en el siguiente código:

```
> fwe <- fitdist(data, "weibull")
> summary(fwe)
Fitting of the distribution ' weibull ' by maximum likelihood
Parameters :
       estimate  Std. Error
shape 94.507833 5.606358842
scale  1.259385 0.001419776
Loglikelihood:  294.4334   AIC:  -584.8667   BIC:  -579.6564
Correlation matrix:
          shape     scale
shape 1.0000000 0.3476946
scale 0.3476946 1.0000000

> fln <- fitdist(data, "lnorm")
> summary(fln)
Fitting of the distribution ' lnorm ' by maximum likelihood
Parameters :
           estimate   Std. Error
meanlog 0.226627519 0.0006956688
sdlog   0.006956688 0.0004469180
Loglikelihood:  332.2486   AIC:  -660.4972   BIC:  -655.2868
Correlation matrix:
             meanlog        sdlog
meanlog 1.000000e+00 4.418253e-15
sdlog   4.418253e-15 1.000000e+00

> fex <- fitdist(data, "exp")
> summary(fex)
Fitting of the distribution ' exp ' by maximum likelihood
Parameters :
      estimate Std. Error
rate 0.7971983 0.07971971
Loglikelihood:  -122.6652   AIC:  247.3304   BIC:  249.9355

> fga <- fitdist(data, "gamma")
> summary(fga)
Fitting of the distribution ' gamma ' by maximum likelihood
Parameters :
      estimate Std. Error
shape 20574.01   5981.308
rate  16401.57   4768.365
Loglikelihood:  332.0381   AIC:  -660.0761   BIC:  -654.8658
Correlation matrix:
          shape      rate
shape 1.0000000 0.9999971
rate  0.9999971 1.0000000
```

- El ***summary*** para el caso del Jellyfish se muestra a continuación:

```
> fwe <- fitdist(data, "weibull")
> summary(fwe)
Fitting of the distribution ' weibull ' by maximum likelihood
Parameters :
       estimate Std. Error
shape 52.945379  3.4877511
scale  7.857702  0.0158298
Loglikelihood:  47.92782   AIC:  -91.85565   BIC:  -86.64531
Correlation matrix:
          shape     scale
```

```
shape 1.0000000 0.3478833
scale 0.3478833 1.0000000

> fln <- fitdist(data, "lnorm")
> summary(fln)
Fitting of the distribution ' lnorm ' by maximum likelihood
Parameters :
          estimate  Std. Error
meanlog 2.05324531 0.001474755
sdlog   0.01474755 0.001021314
Loglikelihood:  74.44944   AIC:  -144.8989   BIC:  -139.6885
Correlation matrix:
              meanlog         sdlog
meanlog  1.000000e+00 -1.070211e-14
sdlog   -1.070211e-14  1.000000e+00

> fex <- fitdist(data, "exp")
> summary(fex)
Fitting of the distribution ' exp ' by maximum likelihood
Parameters :
      estimate Std. Error
rate 0.1283037  0.0128296
Loglikelihood:  -305.3355   AIC:  612.671   BIC:  615.2761

> fga <- fitdist(data, "gamma")
> summary(fga)
Fitting of the distribution ' gamma ' by maximum likelihood
Parameters :
       estimate Std. Error
shape 4565.1834  648.58656
rate   585.7306   83.22066
Loglikelihood:  74.09433   AIC:  -144.1887   BIC:  -138.9783
Correlation matrix:
          shape      rate
shape 1.0000000 0.9999457
rate  0.9999457 1.0000000
```

Básicamente lo que se muestra es un resumen con los parámetros que se deben utilizar en caso de escoger una distribución u otra. Ahora mismo se podría simular una red de colas con todas esas distribuciones ya que tenemos los elementos necesarios. Pero existe una manera, para ser más precisos en la elección y centrarnos en la distribución que realmente nos interesa y es empleando la función *gofstat.*

Mediante esta función, se va a ser capaz de encontrar la distribución que se va a emplear en las simulaciones. Vamos a ver las estadísticas de bondad como se ha explicado en la memoria. Y se elegirá aquella que la distancia sea la mínima. A continuación vemos el *gofstat* de cada uno de los algoritmos.

- Para HotSpot se muestran los siguientes datos:

```
> gofstat(list(fwe, fln, fex, fga), fitnames = c("weibull", "lnorm",
"exp", "gamma"))
Goodness-of-fit statistics
                                 weibull     lnorm        exp     gamma
Kolmogorov-Smirnov statistic 0.4021288 0.4073817  0.6057929 0.3971113
Cramer-von Mises statistic   1.6789666 1.7772569  4.6693319 1.6431820
Anderson-Darling statistic   9.0646006 9.5149489 21.3833537 8.9192327
```

```
Goodness-of-fit criteria
                                 weibull     lnorm      exp     gamma
Aikake's Information Criterion -446.9793 -441.2402 -231.357 -448.8681
Bayesian Information Criterion -440.1552 -437.4161 -229.445 -445.0440
```

- Para Georef los datos obtenidos son los siguientes:

```
> gofstat(list(fwe, fln, fex, fga), fitnames = c("weibull", "lnorm",
"exp", "gamma"))
Goodness-of-fit statistics
                               weibull     lnorm        exp     gamma
Kolmogorov-Smirnov statistic 0.2132438 0.1827710  0.6177045 0.1846999
Cramer-von Mises statistic   1.7435016 0.8798901  9.5419635 0.9058369
Anderson-Darling statistic   9.2979318 5.0030904 43.5527706 5.1345081

Goodness-of-fit criteria
                                 weibull     lnorm       exp     gamma
Aikake's Information Criterion -1495.559 -1552.467 -930.5275 -1550.927
Bayesian Information Criterion -1490.349 -1547.257 -927.9223 -1545.716
```

- Para Fusion se obtienen los datos mostrados a continuación:

```
> gofstat(list(fwe, fln, fex, fga), fitnames = c("weibull", "lnorm",
"exp", "gamma"))
Goodness-of-fit statistics
                               weibull     lnorm        exp     gamma
Kolmogorov-Smirnov statistic  0.2170623 0.1164107  0.6270538 0.1170654
Cramer-von Mises statistic    1.6847868 0.2264718  9.9503833 0.2303810
Anderson-Darling statistic   10.0617953 1.9111081 45.3146847 1.9386090

Goodness-of-fit criteria
                                 weibull     lnorm      exp     gamma
Aikake's Information Criterion -584.8667 -660.4972 247.3304 -660.0761
Bayesian Information Criterion -579.6564 -655.2868 249.9355 -654.8658
```

- Para el caso de Jellyfish los datos obtenidos son los siguientes:

```
> gofstat(list(fwe, fln, fex, fga), fitnames = c("weibull", "lnorm",
"exp", "gamma"))
Goodness-of-fit statistics
                               weibull     lnorm        exp     gamma
Kolmogorov-Smirnov statistic 0.2194776 0.1615305  0.6235832 0.1621912
Cramer-von Mises statistic   1.4014164 0.6485065  9.7985894 0.6579297
Anderson-Darling statistic   8.0692013 3.8173837 44.6608467 3.8723713

Goodness-of-fit criteria
                                 weibull     lnorm      exp     gamma
Aikake's Information Criterion -91.85565 -144.8989 612.6710 -144.1887
Bayesian Information Criterion -86.64531 -139.6885 615.2761 -138.9783
```

Otros parámetros a tener en cuenta son el AIC y BIC que son criterios clásicos basados en la probabilidad logarítmica. En caso de duda puede ser útil para decantarnos por una distribución.

En la *tabla 3.1* que se muestra en el capítulo 3 de la memoria se puede ver cuál ha sido la distribución elegida y que parámetros son los que hay que tener en cuenta para la posterior simulación.

ANEXO 2. GRÁFICOS DEL COMPORTAMIENTO DE CADA ALGORITMO EMPLEANDO DISTRIBUCIONES EXPONENCIALES Y GENERALES.

Este punto del anexo hace referencia al capítulo 4 de la memoria. Concretamente, se va a mostrar el gráfico obtenido de cada algoritmo con la intención de mostrar de manera visual los resultados imputados en la memoria.

2.1 Gráficos empleando distribuciones exponenciales

En este punto del anexo, se va a mostrar gráficamente los resultados obtenidos después de recoger los valores generados por la simulación efectuada en la herramienta OMNeT++. Como se ha ido anotando a lo largo del proyecto, estos resultados se han cargado en R mediante un fichero CSV, que después de un código empleado para generar las gráficas, los resultados utilizando un modelo de colas exponencial son los siguientes:

- Para la Fusion con un modelo de colas exponencial y cuatro servidores:

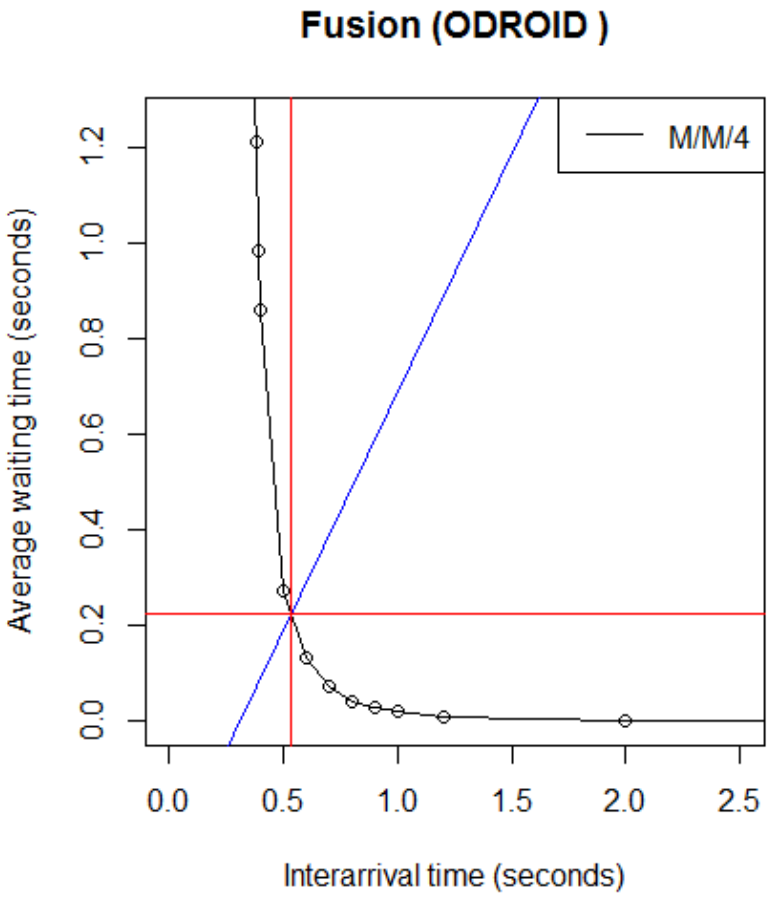

Figura A2.1 Gráfico del Algoritmo Fusion con un modelo de colas exponencial

El procedimiento que se sigue una vez obtenido estos resultados se ejecuta en el capítulo 4 de la memoria. No obstante, anotar que el valor que nos interesa encontrar aquí es el del tiempo entre llegadas de las imágenes en el que su valor es de 0.535 segundos, que viene marcado por el punto de corte entre las dos líneas rojas coincidiendo con la línea azul, que marca el mínimo tiempo entre llegadas.

- Para Jellyfish con un modelo de colas exponencial y ocho servidores definidos, el gráfico obtenido es el siguiente:

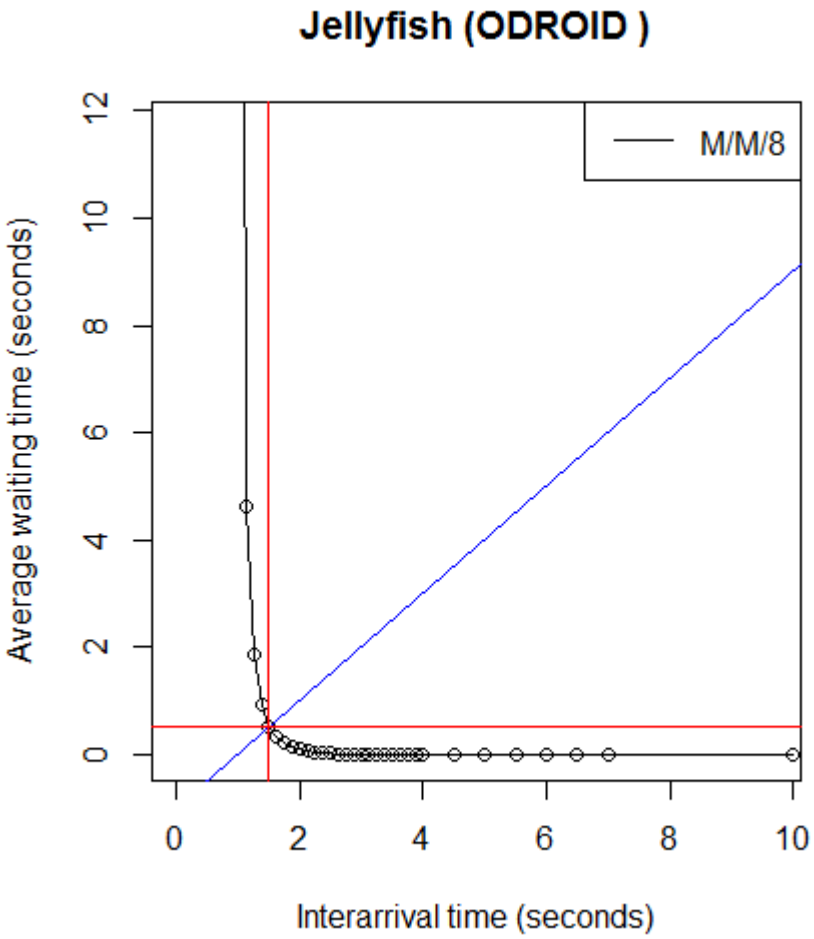

Figura A2.2 Gráfico del Algoritmo Jellyfish con un modelo de colas exponencial

Del mismo modo que en el caso anterior, lo que nos interesa ver en este gráfico para ejecutar la simulación llevando al sistema al límite, es el tiempo entre llegadas mínimo que se produce cuando empleamos ocho servidores utilizando un modelo de colas exponencial. Anotar entonces que el valor de dicho tiempo es de 1.5 segundos.

Quality sería el otro algoritmo que utiliza más de un servidor, pero este se puede ver en la memoria. Para el resto de algoritmos como solo se les dedica un servidor, para el modelo de colas exponencial los gráficos son los utilizados en el capítulo 2 de la memoria.

2.2 Gráficos empleando distribuciones generales

Igual que en el punto anterior se han mostrado los gráficos de los algoritmos utilizando un modelo de colas exponencial, aquí se va a ver el comportamiento de los algoritmos utilizando un modelo de colas determinista. En este caso, sin embargo, se mostrarán todos los algoritmos tengan uno o más servidores, excepto el caso del Quality que se muestra en el capítulo 4 de la memoria.

- Para el caso de HotSpot con un servidor utilizando un modelo de colas con una distribución gamma, el gráfico es el siguiente:

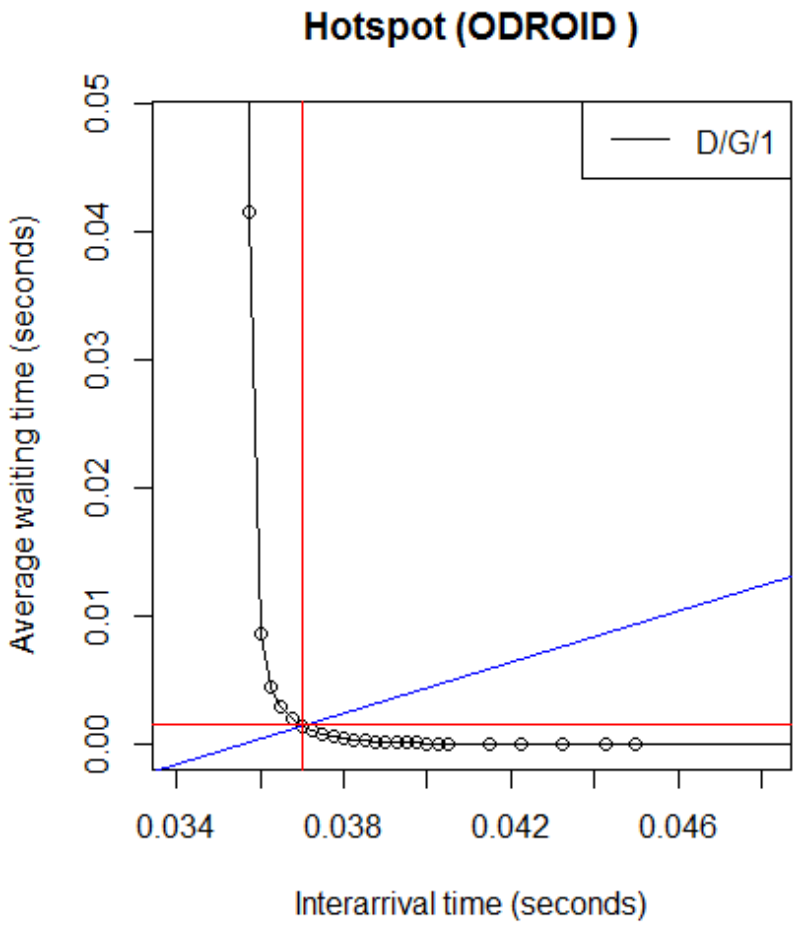

Figura A2.3 Gráfico del Algoritmo HotSpot utilizando una distribución gamma

De esta gráfica podemos extraer que el tiempo mínimo entre llegadas de las imágenes debe ser de 0.037 segundos, como marca el punto de corte encontrado.

- Para el caso de Georef también se utiliza un servidor, que utilizando el modelo de colas con una distribución lognormal, el resultado es el siguiente:

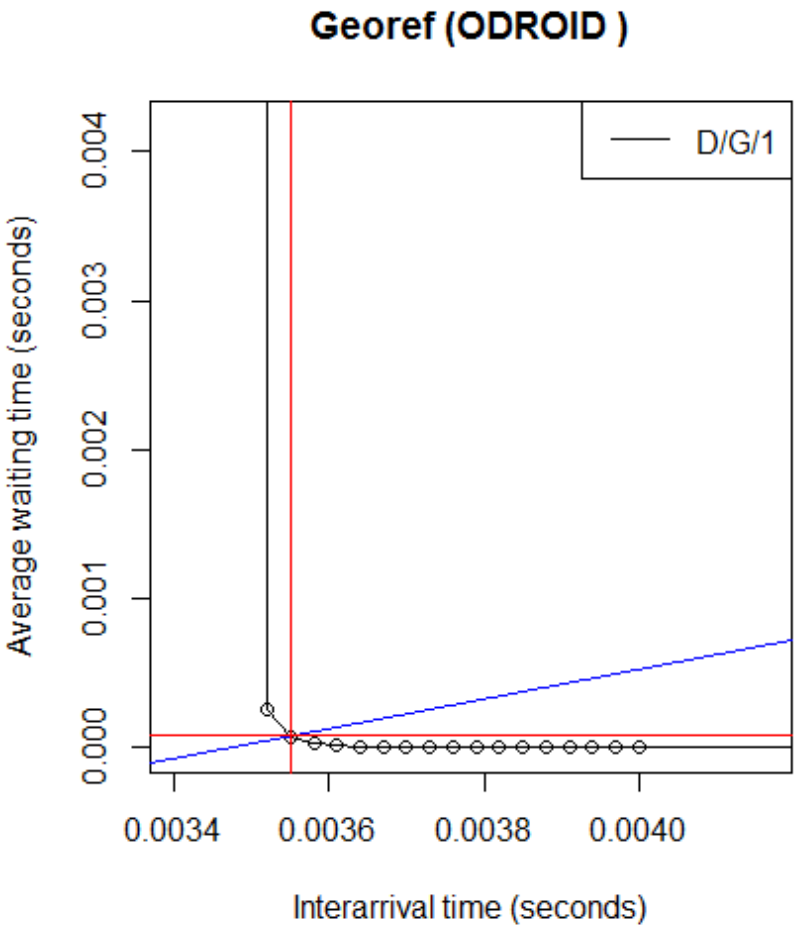

Figura A2.4 Gráfico del Algoritmo Georef utilizando una distribución lognormal

Del mismo modo que en casos anteriores el tiempo de interés en este gráfico es el tiempo mínimo entre llegadas, que su valor es de 0.00355 segundos.

- Para el caso de Fusion se le han distribuido cuatro servidores y el modelo de colas que emplea viene dado por la distribución lognormal:

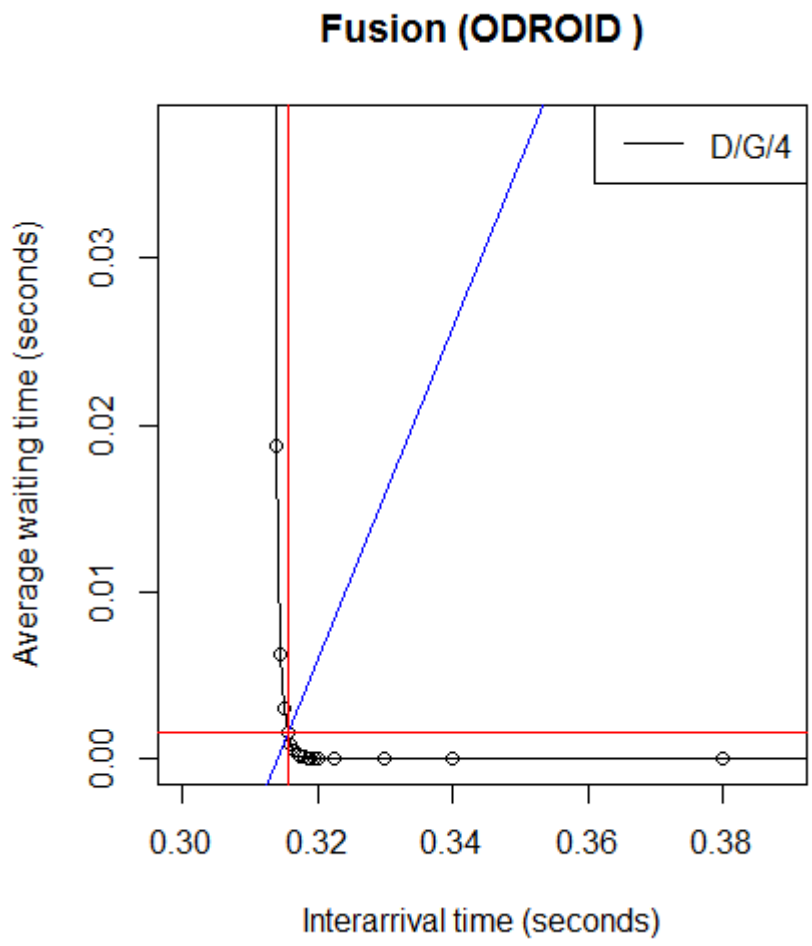

Figura A2.5 Gráfico del Algoritmo Fusion utilizando una distribución lognormal

El tiempo mínimo entre llegadas que marca la Fusion es de 0.3155 segundos. En el caso de uso de HotSpot será este tiempo el que limite el sistema al ser el valor más grande entre todos los algoritmos que componen la red.

- Por último, vemos el caso de Jellyfish, que se le dedican los ocho servidores. Su modelo de colas utiliza una distribución lognormal.

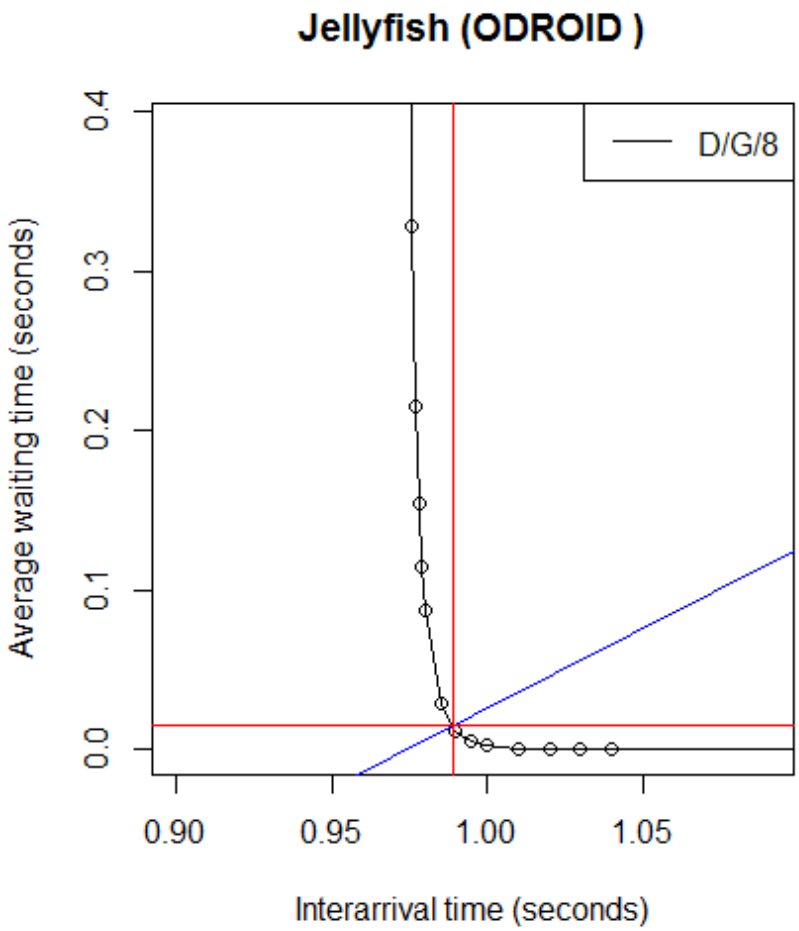

Figura A2.6 Gráfico del Algoritmo Jellyfish utilizando una distribución lognormal

Existe un apartado en la memoria donde se analiza el caso de uso de Jellysifsh. Por lo que el tiempo límite ente llegadas de los paquetes será el que límite el mismo sistema. Como se puede ver en el gráfico el punto de corte entre rectas indica este tiempo, que su valor es de 0.989 segundos.

BIBLIOGRAFIA

[1] Peña, D,. “Fundamentos de estadística”, *Alianza editorial*, Edición electrónica, 2014

[2] Forest, N.V., “Simulating Queueing Networks with OMNeT++”, January 24, 2003

[3] Capitulo 2- Teoría de colas o líneas de espera, *http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/garduno_a_f/capitulo2.pdf*, [Online]

[4] Delignette-Muller, M.L. and Dutang, C., “fitdistrplus: An R Package for Fitting Distributions”, *Journal of Statistical Software*, 64 (4), 1-11(2015).

[5] Ricci, V., “Fitting distributions with R”, Release 0.4-21 February 2005

[6] Capítulo 3- Simulador OMNeT++, http://upcommons.upc.edu/pfc/bitstream/2099.1/5386/4/Levy_Bunan_mem%C3%B2ria_3.pdf [Online]

[7] Capítulo 4- Simulación en OMNeT++, http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/deschamps_e_me/capitulo4.pdf, [Online]

[8] OMNeT++ Simulation Library 5.0, https://omnetpp.org/doc/omnetpp/api/group__RandomNumbersCont.html [Online]

[9] Salamí, E. Barrado, C. Pastor, P and Santamaria, E., “Real-Time Data Processing for the Airborne Detection of Hot Spots”, *Journal of Aerospace Information Systems*, 10 (4), October 2013

[10] Barrado, C. Fuentes, JA. Salamí, E. Royo, P. Olariaga, AD. López, J. Fuentes, VL. Gili, JM. Pastor, E., “Jellyfish monitoring on coastlines using remote piloted aircraft”, *Earth and environmental Science,* 17, 2014

[11] Paradis, E. “R para principiantes”, *Institut des Sciences de l’Évolution*, Marzo 2003

[12] Lee, J.: Odroid-xu3: The fastest computer made by hardkernel so far! ODROID Mag-azine pp. 22–23 (2014)